Abstract machines for game semantics, revisited[*]

Olle Fredriksson and Dan R. Ghica

University of Birmingham, UK

April 16, 2013

Abstract

We define new abstract machines for game semantics which correspond to networks of conventional computers, and can be used as an intermediate representation for compilation targeting distributed systems. This is achieved in two steps. First we introduce the HRAM, a *Heap and Register Abstract Machine*, an abstraction of a conventional computer, which can be structured into HRAM nets, an abstract point-to-point network model. HRAMs are multi-threaded and subsume communication by tokens (*cf.* IAM) or jumps. Game Abstract Machines (GAM), are HRAMs with additional structure at the interface level, but no special operational capabilities. We show that GAMs cannot be naively composed, but composition must be mediated using appropriate HRAM combinators. HRAMs are flexible enough to allow the representation of game models for languages with state (non-innocent games) or concurrency (non-alternating games). We illustrate the potential of this technique by implementing a toy distributed compiler for ICA, a higher-order programming language with shared state concurrency, thus significantly extending our previous distributed PCF compiler. We show that compilation is sound and memory-safe, i.e. no (distributed or local) garbage collection is necessary.

1 Introduction

One of the most profound discoveries in theoretical computer science is the fact that logical and computational phenomena can be subsumed by relatively simple communication protocols. This understanding came independently from Girard's work on the Geometry of Interaction (GOI) [16] and Milner's work on process calculi [22], and had a profound influence on the subsequent development of game semantics (see [12] for a historical survey). Of the three, game semantics proved to be particularly effective at producing precise mathematical models for a large variety of programming languages, solving a long-standing open problem concerning higher-order sequential computation [1, 19].

[*]An extended abstract of this paper is due to appear in the Twenty-Eighth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013), June 25-28, 2013, New Orleans, USA.

One of the most appealing features of game semantics is that it has a dual denotational and operational character. By *denotational* we mean that it is compositionally defined on the syntax and by *operational* we mean that it can be effectively presented and can form a basis for compilation [13]. This feature was apparent from the earliest presentations of game semantics [18] and is not very surprising, although the operational aspects are less perspicuous than in interpretations based on process calculi or GOI, which quickly found applications in compiler [21] or interpreter [2] development and optimisation.

An important development, which provided essential inspiration for this work, was the introduction of the *Pointer Abstract Machine* (PAM) and the *Interaction Abstract Machine* (IAM), which sought to fully restore the operational intuitions of game semantics [5] by relating them to two kinds of abstract machines, one based on term rewriting (PAM) and one based on networks of automata (IAM) profoundly inspired by GOI. A further optimisation of IAM, the *Jumping Abstract Machine* (JAM) was introduced subsequently to avoid the overheads of the IAM [6].

Contribution In this paper we are developing the line of work on the PAM, IAM, and JAM, in order to define new abstract machines which correspond more closely to *networks of conventional computers* and can be used as an intermediate representation for compilation targeting distributed systems. This is achieved in two steps. First we introduce the HRAM, a *Heap and Register Abstract Machine*, an abstraction of a conventional computer, which can be structured into HRAM nets, an abstract point-to-point network model. HRAMs are multi-threaded and subsume communication by tokens (cf. IAM) or jumps. GAMs, *Game Abstract Machines*, are HRAMs with additional structure at the interface level, but no special operational capabilities. We show that GAMs cannot be naively composed, but composition must be mediated using appropriate HRAM combinators. Starting from a formulation of game semantics in the nominal model [9] has two benefits. First, pointer manipulation requires no encoding or decoding, as in integer-based representations, but exploits the HRAM ability to create locally fresh *names*. Second, token size is constant as only names are passed around; the computational history of a token is stored by the HRAM rather than passing it around (cf. IAM). HRAMs are also flexible enough to allow the representation of game models for languages with state (*non-innocent* games) or concurrency (*non-alternating* games). We illustrate the potential of this technique by implementing a compiler targeting distributed systems for ICA, a higher-order programming language with shared state concurrency [14], thus significantly extending our previous distributed PCF compiler [8]. We show that compilation is sound and memory-safe, i.e. no (distributed or local) garbage collection is necessary.[1]

Other related and relevant work The operational intuitions of GOI were originally confined to the sequential setting, but more recent work on Ludics showed how they can be applied to concurrency [7] through an abstract treatment not immediately applicable to our needs. Whereas our work takes the IAM/JAM as the starting point, developing abstract machines akin to the PAM

[1]Available from `http://veritygos.org/gams`.

revealed interesting syntactic and operational connections between game semantics and Böhm trees [4]. The connection between game semantics, syntactic recursion schemes and automata also had several interesting applications to verifying higher-order computation (see e.g. [24]). Finally, the connection between game semantics and operational semantics can be made more directly by eliding all the semantic structure in the game and reducing them to a very simple communication mechanism between a program and its environment, which is useful in understanding hostile opponents and verifying security properties [15].

2 Simple nets

In this section we introduce a class of basic abstract machines for manipulating heap structures, which also have primitives for communications and control. They represent a natural intermediate stage for compilation to machine language, and will be used as such in Sec. 4. The machines can naturally be organised into communication networks which give an abstract representation of distributed systems. We find it formally convenient to work in a nominal model in order to avoid the difficulties caused by concrete encoding of game structures, especially *justification pointers*, as integers. We assume a certain familiarity from the reader with basic nominal concepts. The interested reader is referred to the literature ([10] is a starting point).

2.1 Heap and register abstract machines (HRAM)

We fix a set of *port names* ($\mathbb{A}$) and a set of *pointer names* ($\mathbb{P}$) as disjoint sets of atoms. Let $L \triangleq \{\mathbf{O}, \mathbf{P}\}$ be the set of polarities of a port. To maintain an analogy with game semantics from the beginning, port names correspond to game-semantic *moves* and input/output polarities correspond to opponent/proponent. A *port structure* is a tuple $(l, a) \in \mathcal{Port} = L \times \mathbb{A}$. An *interface* $A \in \mathcal{P}_{fin}(\mathcal{Port})$ is a set of port structures such that all port names are unique, i.e. $\forall p = (l, a), p' = (l', a') \in A$, if $a = a'$ then $p = p'$. Let the support of an interface be $sup(A) \triangleq \{a \mid (l, a) \in A\}$, its set of port names.

The *tensor* of two interfaces is defined as $A \otimes B \triangleq A \cup B$, where $sup(A) \cap sup(B) = \emptyset$. The dual of an interface is defined as $A^* \triangleq \{p^* \mid p \in A\}$ where $(l, a)^* \triangleq (l^*, a)$, $\mathbf{O}^* \triangleq \mathbf{P}$ and $\mathbf{P}^* \triangleq \mathbf{O}$. An arrow interface is defined in terms of tensor and dual, $A \Rightarrow B \triangleq A^* \otimes B$.

We introduce notation for opponent ports of an interface $A^{(\mathbf{O})} \triangleq \{(\mathbf{O}, a) \in A\}$. The player ports of an interface $A^{(\mathbf{P})}$ is defined analogously. The set of all interfaces is denoted by $\mathcal{I}$. We say that two interfaces *have the same shape* if they are equivariant, i.e. there is a permutation $\pi : \mathbb{A} \to \mathbb{A}$ such that $\{\pi \cdot p \mid p \in A_1\} = A_2$, and we write $\pi \vdash A_1 =_{\mathbb{A}} A_2$, where $\pi \cdot (l, a) \triangleq (l, \pi(a))$ is the permutation action of π. We may only write $A_1 =_{\mathbb{A}} A_2$ if π is obvious or unimportant.

Let the set of data $\mathcal{D}$ be $\emptyset \in \mathbb{1}$, pointer names $a \in \mathbb{P}$ or integers $n \in \mathbb{Z}$. Let the set of instructions Instr be as below, where $i, j, k \in \mathbb{N} + \mathbb{1}$ (which permits ignoring results and allocating "null" data).

- $i \leftarrow$ `new` j, k allocates a new pointer in the heap and populates it with the values stored in registers j and k, storing the pointer in register i.
- $i, j \leftarrow$ `get` k reads the tuple pointed at by the name in the register k and stores it in registers i and j.
- `update` i, j writes the value stored in register j to the second component of the value pointed to by the name in register i.
- `free` i releases the memory pointed to by the name in the register i and resets the register.
- `flip` i, j flips the values of registers i and j.
- $i \leftarrow$ `set` j sets register i to value j.

Let code fragments $\mathcal{C}$ be $\mathcal{C} ::= \mathit{Instr}; \mathcal{C} \mid$ `ifzero` $\mathbb{N}\ \mathcal{C}\ \mathcal{C} \mid$ `spark` $a \mid$ `end`. The port names occurring in the code fragment are $\mathit{sup} \in \mathcal{C} \to \mathcal{P}_{\mathit{fin}}(\mathbb{A})$, defined in the obvious way (only the `spark` a instruction can contribute names). An `ifzero` i instruction will branch according to the value stored in register i. A `spark` a will either jump to a or send a message to a, depending on whether a is a local port or not.

An *engine* is an interface together with a port map, $E = (A, P) \in \mathcal{I} \times (\mathit{sup}(A^{(\mathbf{O})}) \to \mathcal{C})$ such that for each code fragment $c \in \operatorname{cod} P$ and each port name $a \in \mathit{sup}(c)$, $(\mathbf{P}, a) \in A$, meaning that ports that are "sparked" must be output ports of the interface A. The set of all engines is $\mathcal{E}$.

Engines have threads and shared heap. All threads have a fixed number of registers r, which is a global constant. For the language ICA we will need four registers, but languages with more kinds of pointers in the game model, e.g. control pointers [20], may need and use more registers.

A *thread* is a tuple $t = (c, \overline{d}) \in T = \mathcal{C} \times \mathcal{D}^r$: a code fragment and an r-tuple of data register values.

An *engine configuration* is a tuple $k = (\overline{t}, h) \in \mathcal{K} = \mathcal{P}_{\mathit{fin}}(T) \times (\mathbb{P} \rightharpoonup \mathbb{P} \times \mathcal{D})$: a set of threads and a heap that maps pointer names to pairs of pointer names and data items.

A pair consisting of an engine configuration and an engine will be written using the notation $k : E \in \mathcal{K} \times \mathcal{E}$. Define the function $\mathit{initial} \in \mathcal{E} \to \mathcal{K} \times \mathcal{E}$ as $\mathit{initial}(E) \triangleq (\emptyset, \emptyset) : E$ for an engine E. This function pairs the engine up with an engine configuration consisting of no threads and an empty heap.

HRAMs communicate using *messages*, each consisting of a port name and a vector of data items of size r_m: $m = (x, \overline{d}) \in \mathcal{M} = \mathbb{A} \times \mathcal{D}^{r_m}$. The constant r_m specifies the size of the messages in the network, and has to fulfil $r_m \leq r$. For a set $X \subseteq \mathbb{A}$, define $\mathcal{M}_X = X \times \mathcal{D}^{r_m}$, the subset of $\mathcal{M}$ whose port names are limited to those of X.

We specify the operational semantics of an engine $E = (A, P)$ as a transition relation $- \xrightarrow[E,\chi]{-} - \subseteq \mathcal{K} \times (\{\bullet\} \cup (L \times \mathcal{M})) \times \mathcal{K}$. The relation is either labelled with

$\bullet$ — a silent transition — or a polarised message — an observable transition. The messages will be constructed simply from the first r_m registers of a thread, meaning that on certain actions part of the register contents become observable in the transition relation.

To aid readability, we use the following shorthands:

- $n \xrightarrow[E,\chi]{} n'$ means $n \xrightarrow[E,\chi]{\bullet} n'$ (silent transitions).
- $n \xrightarrow[E,\chi]{(a,\overline{d})} n'$ means $n \xrightarrow[E,\chi]{(\mathbf{P},(a,\overline{d}))} n'$ (output transitions).
- $n \xrightarrow[E,\chi]{(a,\overline{d})^{\bullet}} n'$ means $n \xrightarrow[E,\chi]{(\mathbf{O},(a,\overline{d}))} n'$ (input transitions).

We use the notation $\overline{d}$ for n-tuples of registers and then d_i for the (zero-based) i-th component of $\overline{d}$, and $d_\emptyset \triangleq \emptyset$. For updating a register, we use $\overline{d}[i := d] \triangleq (d_0, \cdots, d_{i-1}, d, d_{i+1}, \cdots, d_{n-1})$ and $\overline{d}[\emptyset := d] \triangleq \overline{d}$.

To construct messages from the register contents of a thread, we use the functions $msg \in \mathcal{D}^r \to \mathcal{D}^{r_m}$, which takes the first r_m components of its input, and $regs \in \mathcal{D}^{r_m} \to \mathcal{D}^r$, which pads its input with $\emptyset$ at the end (i.e. $regs(\overline{d}) \triangleq (d_0, \ldots, d_{r_m-1}, \emptyset, \ldots)$).

The network connectivity is specified by the function χ, which will be described in more detail in the next sub-section. For a port name a, $\chi(a)$ can be read as "the port that a is connected to". The full operational rules for HRAMs are given in Fig. 2.1. The interesting rule is that for `spark` because it depends on whether the port where the next computation is "sparked" is local or not. If the port is local then `spark` makes a jump, and if the port is non-local then it produces an output token and the current thread of execution is terminated, similar to the IAM.

2.2 HRAM nets

A well-formed *HRAM net* $S \in \mathcal{S}$ is a set of engines, a function over port names specifying what ports are connected, and an external interface, $S = (\overline{E}, \chi, A)$, where $E \in \mathcal{E}, A \in \mathcal{I}$, and χ is a bijection between the net's output and input port names. Specifically, χ has to be in $sup(A^{(\mathbf{O})} \otimes A_{\overline{E}}^{(\mathbf{P})}) \to sup(A^{(\mathbf{P})} \otimes A_{\overline{E}}^{(\mathbf{O})})$, where $A_{\overline{E}} = \otimes\{A \mid (A, P) \in \overline{E}\}$.

Fig. 2 shows a diagram of an HRAM net with two HRAMs (interfaces A, A', two ports each), each with two running threads (t_i, t'_i) with local registers (d_i, d'_i) and shared heaps (h, h'). Two of the HRAM ports are connected and two are part of the global interface B.

The function χ gives the net connectivity. It being in $sup(A^{(\mathbf{O})} \otimes A_{\overline{E}}^{(\mathbf{P})}) \to sup(A^{(\mathbf{P})} \otimes A_{\overline{E}}^{(\mathbf{O})})$ means that it maps each input port name of the net's interface and output port name of the net's engines to either an output port name of the net's interface or an input port name of one of its engines. Since it is a bijection,

$$((i \leftarrow \texttt{new}\ j,k;\ C,\overline{d}) \cup \overline{t}, h) \xrightarrow[E,\chi]{} ((C,\overline{d}[i := p]) \cup \overline{t}, h \cup \{p \mapsto (d_j, d_k)\}) \text{ if } p \notin \mathit{sup}(h)$$

$$((i,j \leftarrow \texttt{get}\ k;\ C,\overline{d}) \cup \overline{t}, h \cup \{d_k \mapsto (d,d')\}) \xrightarrow[E,\chi]{}$$
$$((C,\overline{d}[i := d][j := d']) \cup \overline{t}, h \cup \{d_k \mapsto (d,d')\})$$

$$((\texttt{update}\ i,j;\ C,\overline{d}) \cup \overline{t}, h \cup \{d_i \mapsto (d,d')\}) \xrightarrow[E,\chi]{}$$
$$((C,\overline{d}[i := d][j := d']) \cup \overline{t}, h \cup \{d_i \mapsto (d,d_j)\})$$

$$((\texttt{free}\ i;\ C,\overline{d}) \cup \overline{t}, h \cup \{d_i \mapsto (d,d')\}) \xrightarrow[E,\chi]{} ((C,\overline{d}[i := \emptyset]) \cup \overline{t}, h)$$

$$((\texttt{flip}\ i,j;\ C,\overline{d}) \cup \overline{t}, h) \xrightarrow[E,\chi]{} ((C,\overline{d}[i := d_j][j := d_j]) \cup \overline{t}, h)$$

$$((i \leftarrow \texttt{set}\ j;\ C,\overline{d}) \cup \overline{t}, h) \xrightarrow[E,\chi]{} ((C,\overline{d}[i := j]) \cup \overline{t}, h)$$

$$((\texttt{ifzero}\ i\ c_1\ c_2;\ C,\overline{d}[i := 0]) \cup \overline{t}, h) \xrightarrow[E,\chi]{} ((c_1,\overline{d}[i := \emptyset]) \cup \overline{t}, h)$$

$$((\texttt{ifzero}\ i\ c_1\ c_2;\ C,\overline{d}[i := n+1]) \cup \overline{t}, h) \xrightarrow[E,\chi]{} ((c_2,\overline{d}[i := \emptyset]) \cup \overline{t}, h)$$

$$((\texttt{spark}\ a,\overline{d}) \cup \overline{t}, h) \xrightarrow[E,\chi]{(\chi(a), msg(\overline{d}))} (\overline{t}, h) \text{ if } (\mathbf{O}, \chi(a)) \notin A$$

$$((\texttt{spark}\ a,\overline{d}) \cup \overline{t}, h) \xrightarrow[E,\chi]{} ((P(\chi(a)), \mathit{regs}(msg(\overline{d}))) \cup \overline{t}, h) \text{ if } (\mathbf{O}, \chi(a)) \in A$$

$$(\overline{t}, h) \xrightarrow[E,\chi]{(a,\overline{d})^{\bullet}} ((P(a), \mathit{regs}(\overline{d})) \cup \overline{t}, h) \text{ if } (\mathbf{O}, a) \in A$$

$$((\texttt{end}, \overline{d}) \cup \overline{t}, h) \xrightarrow[E,\chi]{} (\overline{t}, h)$$

Figure 1: Operational semantics of HRAMs

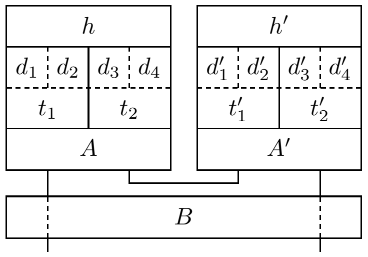

Figure 2: Example HRAM net

$$\frac{e \xrightarrow[E,\chi]{} e'}{(e\ :\ E \cup \overline{e\ :\ E}, \overline{m}) \to (e'\ :\ E \cup \overline{e\ :\ E}, \overline{m})}$$

$$\frac{e \xrightarrow[E,\chi]{m} e'}{(e\ :\ E \cup \overline{e\ :\ E}, \overline{m}) \to (e'\ :\ E \cup \overline{e\ :\ E}, \{m\} \uplus \overline{m})}$$

$$\frac{e \xrightarrow[E,\chi]{m^{\bullet}} e'}{(e\ :\ E \cup \overline{e\ :\ E}, \{m\} \uplus \overline{m}) \to (e'\ :\ E \cup \overline{e\ :\ E}, \overline{m})}$$

$$\frac{(\mathbf{P}, a) \in A}{(\overline{e\ :\ E}, \{(a, \overline{d})\} \uplus \overline{m}) \xrightarrow{(a,\overline{d})} (\overline{e\ :\ E}, \overline{m})}$$

$$\frac{(\mathbf{O}, a) \in A}{(\overline{e\ :\ E}, \overline{m}) \xrightarrow{(a,\overline{d})^{\bullet}} (\overline{e\ :\ E}, \{(\chi(a), \overline{d})\} \uplus \overline{m})}$$

Figure 3: Operational semantics of HRAM nets

each port name (and thus port) is connected to exactly one other port name, so the abstract network model we are using is point-to-point.

For an engine $e = (A, P)$, we define a *singleton* net with e as its sole engine as *singleton*$(e) = (\{e\}, \chi, A')$, where A' is an interface such that $\chi \vdash A =_{\mathbb{A}} A'$ and χ is given by:

$$\chi(a) \triangleq \pi(a) \text{ if } a \in \mathit{sup}(A^{(\mathbf{P})})$$
$$\chi(a) \triangleq \pi^{-1}(a) \text{ if } a \in \mathit{sup}(A'^{(\mathbf{O})})$$

A *net configuration* is a set of tuples of engine configurations and engines and a multiset of pending messages: $n = (\overline{e\ :\ E}, \overline{m}) \in \mathcal{N} = \mathcal{P}_{\mathit{fin}}(\mathcal{K} \times \mathcal{E}) \times \mathbf{Mset}_{\mathit{fin}}(\mathcal{M})$. Define the function *initial* $\in \mathcal{S} \to \mathcal{N}$ as $\mathit{initial}(\overline{E}, \chi, A) \triangleq (\{\mathit{initial}(E) \mid E \in \overline{E}\}, \emptyset)$, a net configuration with only initial engines and no pending messages.

The operational semantics of a net $S = (\overline{E}, \chi, A)$ is specified as a transition relation $- \xrightarrow{-} - \subseteq \mathcal{N} \times (\{\bullet\} \cup (L \times \mathcal{M}_{\mathit{sup}(A)})) \times \mathcal{N}$. The semantics is given in the style of the Chemical Abstract Machine (CHAM) [3], where HRAMs are "molecules" and the pending messages of the HRAM net is the "solution". HRAM inputs (outputs) are to (from) the set of pending messages. Silent transitions of any HRAM are silent transitions of the net. The rules are given in Fig. 2.2.

2.3 Semantics of HRAM nets

We define **List**$[A]$ for a set A to be finite sequences of elements from A, and use $s{::}s'$ for concatenation. A *trace* for a net $(\overline{E}, \chi, A)$ is a *finite sequence* of messages with polarity: $s \in \mathbf{List}[L \times \mathcal{M}_{sup(A)}]$. Write $\alpha \in L \times \mathcal{M}_{sup(A)}$ for single polarised messages. We use the same notational convention as before to identify inputs $(-^\bullet)$.

For a trace $s = \alpha_1{::}\alpha_2{::}\cdots{::}\alpha_n$, define $\xrightarrow{s}$ to be the following composition of relations on net configurations: $\xrightarrow{\alpha_1}\to^*\xrightarrow{\alpha_2}\to^* \cdots \xrightarrow{\alpha_n}$, where $\to^*$ is the reflexive transitive closure of $\to$, i.e. any number of silent steps are allowed in between those that are observable.

Write traces_A for the set $\mathbf{List}[L \times \mathcal{M}_{sup(A)}]$. The denotation $[\![S]\!] \subseteq \mathit{traces}_A$ of a net $S = (\overline{E}, \chi, A)$ is the set of traces of observable transitions reachable from the initial net configuration $\mathit{initial}(S)$ using the transition relation:

$$[\![S]\!] \stackrel{\Delta}{=} \{s \in \mathit{traces}_A \mid \exists n.\mathit{initial}(S) \xrightarrow{s} n\}$$

The denotation of a net includes the empty trace and is prefix-closed by construction.

As with interfaces, we are not interested in the actual port names occurring in a trace, so we define *equivariance* for sets of traces. Let $S_1 \subseteq \mathit{traces}_{A_1}$ and $S_2 \subseteq \mathit{traces}_{A_2}$ for $A_1, A_2 \in \mathcal{I}$. $S_1 =_{\mathbb{A}} S_2$ if and only if there is a permutation $\pi \in \mathbb{A} \to \mathbb{A}$ such that $\{\pi \cdot s \mid s \in S_1\} = S_2$, where $\pi \cdot \epsilon \stackrel{\Delta}{=} \epsilon$ and $\pi \cdot (s{::}(l, (a, \overline{d}))) \stackrel{\Delta}{=} (\pi \cdot s){::}(l, (\pi(x), \overline{d}))$.

Define the *deletion* operation $s{-}A$ which removes from a trace all elements $(l, (x, \overline{d}))$ if $x \in sup(A)$ and define the interleaving of sets of traces $S_1 \subseteq \mathit{traces}_A$ and $S_2 \subseteq \mathit{traces}_B$ as $S_1 \otimes S_2 \stackrel{\Delta}{=} \{s \mid s \in \mathit{traces}_{A \otimes B} \wedge s{-}B \in S_1 \wedge s{-}A \in S_2\}$.

Define the composition of the sets of traces $S_1 \subseteq \mathit{traces}_{A \Rightarrow B}$ and $S_2 \subseteq \mathit{traces}_{B' \Rightarrow C}$ with $\pi \vdash B =_{\mathbb{A}} B'$ as the usual *synchronisation and hiding* in trace semantics:

$$S_1; S_2 \stackrel{\Delta}{=} \{s{-}B \mid s \in \mathit{traces}_{A \otimes B \otimes C} \wedge s{-}C \in S_1 \wedge \pi \cdot s^{*B}{-}A \in S_2\}$$

(where s^{*B} is s where the messages from B have reversed polarity.)

Two nets, $f = (\overline{E}_f, \chi_f, I_f)$ and $g = (\overline{E}_g, \chi_g, I_g)$ are said to be *structurally equivalent* if they are graph-isomorphic, i.e. $\pi \cdot \overline{E}_f = \overline{E}_g$, $\pi \vdash I_f =_{\mathbb{A}} I_g$ and $\chi_g \circ \pi = \pi \circ \chi_f$.

Theorem 2.1. *If S_1 and S_2 are structurally equivalent nets, then $[\![S_1]\!] =_{\mathbb{A}} [\![S_2]\!]$.*

Proof. A straightforward induction on the trace length, in both directions. □

2.4 HRAM nets as a category

In this sub-section we will show that HRAM nets form a symmetric compact-closed category. This establishes that our definitions are sensible and that HRAM nets are equal up to topological isomorphisms. This result also shows that the structure of HRAM nets is very loose.

The category, called **HRAMnet** , is defined as follows:

- Objects are interfaces $A \in \mathcal{P}_{fin}(\mathcal{Port})$ identified up to $\mathbb{A}$-equivalence.
- A morphism $f : A \to B$ is a well-formed net on the form $(\overline{E}, \chi, A \Rightarrow B)$, for some $\overline{E}$ and χ. We will identify morphisms that have the same denotation, i.e. if $[\![f]\!] =_{\mathbb{A}} [\![g]\!]$ then $f = g$ (in the category).
- The identity morphism for an object A is

$$id_A \triangleq (\emptyset, \chi, A \Rightarrow A')$$

for an A' such that $\pi \vdash A =_{\mathbb{A}} A'$ and

$$\chi(a) \triangleq \pi(a) \text{ if } a \in sup(A^{*(\mathbf{O})})$$
$$\chi(a) \triangleq \pi^{-1}(a) \text{ if } a \in sup(A'^{(\mathbf{O})}).$$

Note that $A \Rightarrow A' = A^* \cup A'$. This means that the identity is pure connectivity.

- Composition of two morphisms $f = (\overline{E}_f, \chi_f, A \Rightarrow B) : A \to B$ and $g = (\overline{E}_g, \chi_g, B' \Rightarrow C) : B' \to C$, such that $\pi \vdash B =_{\mathbb{A}} B'$, is

$$f;g = (\overline{E}_f \cup \overline{E}_g, \chi_{f;g}, A \Rightarrow C) : A \to C$$

where

$$\chi_{f;g}(a) \triangleq \chi_f(a) \text{ if } a \in sup(A^{*(\mathbf{O})} \otimes I_f^{(\mathbf{P})}) \wedge \chi_f(a) \notin sup(B)$$
$$\chi_{f;g}(a) \triangleq \chi_g(a) \text{ if } a \in sup(C^{(\mathbf{O})} \otimes I_g^{(\mathbf{P})}) \wedge \chi_g(a) \notin sup(B')$$
$$\chi_{f;g}(a) \triangleq \chi_g(\pi(\chi_f(a))) \text{ if } a \in sup(A^{*(\mathbf{O})} \otimes I_f^{(\mathbf{P})}) \wedge \chi_f(a) \in sup(B)$$
$$\chi_{f;g}(a) \triangleq \chi_f(\pi^{-1}(\chi_g(a))) \text{ if } a \in sup(C^{(\mathbf{O})} \otimes I_g^{(\mathbf{P})}) \wedge \chi_g(a) \in sup(B')$$

and

$$I_f \triangleq \otimes\{A \mid (A, P) \in \overline{E}_f\}$$
$$I_g \triangleq \otimes\{A \mid (A, P) \in \overline{E}_g\}.$$

Note We identify HRAMs with interfaces of the same shape in the category, which means that our objects and morphisms are in reality unions of equivariant sets. In defining the operations of our category we use *representatives* for these sets, and require that the representatives are chosen such that their sets of port names are disjoint (but same-shaped when the operation calls for it). The composition operation may appear to be partial because of this requirement, but we can always find equivariant representatives that fulfil it.

It is possible to find other representations of interfaces that do not rely on equivariance. For instance, an interface could simply be two natural numbers — the number of input and output ports. Another possibility would be to make the tensor the *disjoint* union operator. Both of these would, however, lead to a lot of bureaucracy relating to injection functions to make sure that port connections are routed correctly. Our formulation, while seemingly complex, leads to very little bureaucracy, and is easy to implement.

Proposition 2.2. ***HRAMnet*** *is a category.*

Proof.

- Composition is well-defined, i.e. it preserves well-formedness.

  Let $f = (\overline{E}_f, \chi_f, A \Rightarrow B) : A \to B$ and $g = (\overline{E}_g, \chi_g, B' \Rightarrow C) : B' \to C$ be morphisms such that $\pi \vdash B =_{\mathbb{A}} B'$, and their composition $f;g = (\overline{E}_f \cup \overline{E}_g, \chi, A \Rightarrow C) : A \to C$ be as in the definition of composition. To prove that this is well-formed, we need to show that

  $$\begin{aligned}\chi \in \mathit{sup}((A \Rightarrow C)^{(\mathbf{O})} \otimes I_{fg}^{(\mathbf{P})}) \to \mathit{sup}((A \Rightarrow C)^{(\mathbf{P})} \otimes I_{fg}^{(\mathbf{O})}) = \\ \mathit{sup}(A^{*(\mathbf{O})} \otimes C^{(\mathbf{O})} \otimes I_f^{(\mathbf{P})} \otimes I_g^{(\mathbf{P})}) \to \mathit{sup}(A^{*(\mathbf{P})} \otimes C^{(\mathbf{O})} \otimes I_f^{(\mathbf{O})} \otimes I_g^{(\mathbf{O})})\end{aligned}$$

  where $I_{fg} = \otimes\{A \mid (A, P) \in \overline{E}_f \cup \overline{E}_g\}$, and that it is a bijection.

  We are given that

  $$\begin{aligned}\chi_f &\in \mathit{sup}(A^{*(\mathbf{O})} \otimes B^{(\mathbf{O})} \otimes I_f^{(\mathbf{P})}) \to \mathit{sup}(A^{*(\mathbf{P})} \otimes B^{(\mathbf{P})} \otimes I_f^{(\mathbf{O})}) \\ \chi_g &\in \mathit{sup}(B'^{*(\mathbf{O})} \otimes C^{(\mathbf{O})} \otimes I_g^{(\mathbf{P})}) \to \mathit{sup}(B'^{*(\mathbf{P})} \otimes C^{(\mathbf{P})} \otimes I_g^{(\mathbf{O})}) \\ \pi &\in \mathit{sup}(B) \to \mathit{sup}(B')\end{aligned}$$

  are bijections.

  It is relatively easy to see that the domains specified in the clauses of the definition of χ are mutually disjoint sets and that their union is the domain that we are after.

  Since χ is defined in clauses each of which defined using either χ_f or χ_g and/or π (which are bijections with disjoint domains and codomains), it is enough to show that the set of port names that χ_f is applied to in clause 1 and 4 are disjoint, and similarly for χ_g in clause 2 and 3:

  – In clause 4, we have $\chi_g(a) \in \mathit{sup}(B')$, and so $\pi^{-1}(\chi_g(a)) \in \mathit{sup}(B)$, which is disjoint from $\mathit{sup}(A^{*(\mathbf{O})} \otimes I_f^{(\mathbf{P})})$ in clause 1.

  – In clause 3, we have $\chi_f(a) \in \mathit{sup}(B)$, and so $\pi(\chi_f(a)) \in \mathit{sup}(B')$, which is disjoint from $\mathit{sup}(C^{(\mathbf{O})} \otimes I_g^{(\mathbf{P})})$ in clause 2.

- Composition is associative.

  Let

  $$\begin{aligned}f &= (\overline{E}_f, \chi_f, A \Rightarrow B) : A \to B, \\ g &= (\overline{E}_g, \chi_g, B' \Rightarrow C) : B' \to C, \text{ and} \\ h &= (\overline{E}_h, \chi_h, C' \Rightarrow D) : C' \to D\end{aligned}$$

  be nets such that $\pi_1 \vdash B =_{\mathbb{A}} B'$ and $\pi_2 \vdash C =_{\mathbb{A}} C'$. Then we have:

  $$(f;g);h = (\overline{E}_f \cup \overline{E}_g \cup \overline{E}_h, \chi_{(f;g);h}, A \Rightarrow D)$$

  and

  $$f;(g;h) = (\overline{E}_f \cup \overline{E}_g \cup \overline{E}_h, \chi_{f;(g;h)}, A \Rightarrow D)$$

  according to the definition of composition. We need to show that $\chi_{(f;g);h} = \chi_{f;(g;h)}$, which implies that $(f;g);h = f;(g;h)$.

We do this by expanding the definitions, simplified using the following auxiliary function:

$$\begin{aligned} \mathit{connect}(c, A)(a) &\triangleq a && \text{if } a \notin \mathit{sup}(A) \\ \mathit{connect}(c, A)(a) &\triangleq c(a) && \text{if } a \in \mathit{sup}(A) \end{aligned}$$

$f; g = (\overline{E}_f \cup \overline{E}_g, \chi_{f;g}, A \Rightarrow C)$ and $g; h = (\overline{E}_g \cup \overline{E}_h, \chi_{g;h}, B' \Rightarrow D)$ where

$$\begin{aligned} \chi_{f;g}(a) &\triangleq \mathit{connect}(\chi_g \circ \pi_1, B)(\chi_f(a)) \text{ if } a \in \mathit{sup}(A^{*(\mathbf{O})} \otimes I_f^{(\mathbf{P})}) \\ \chi_{f;g}(a) &\triangleq \mathit{connect}(\chi_f \circ \pi_1^{-1}, B')(\chi_g(a)) \text{ if } a \in \mathit{sup}(C^{(\mathbf{O})} \otimes I_g^{(\mathbf{P})}) \\ \chi_{g;h}(a) &\triangleq \mathit{connect}(\chi_h \circ \pi_2, C)(\chi_g(a)) \text{ if } a \in \mathit{sup}(B'^{*(\mathbf{O})} \otimes I_g^{(\mathbf{P})}) \\ \chi_{g;h}(a) &\triangleq \mathit{connect}(\chi_g \circ \pi_2^{-1}, C')(\chi_h(a)) \text{ if } a \in \mathit{sup}(D^{(\mathbf{O})} \otimes I_h^{(\mathbf{P})}) \end{aligned}$$

Now $\chi_{(f;g);h}$ and $\chi_{f;(g;h)}$ are defined as follows:

$$\begin{aligned} \chi_{(f;g);h}(a) &\triangleq \mathit{connect}(\chi_h \circ \pi_2, C)(\chi_{f;g}(a)) \text{ if } a \in \mathit{sup}(A^{*(\mathbf{O})} \otimes I_{f;g}^{(\mathbf{P})}) \\ \chi_{(f;g);h}(a) &\triangleq \mathit{connect}(\chi_{f;g} \circ \pi_2^{-1}, C')(\chi_h(a)) \text{ if } a \in \mathit{sup}(D^{(\mathbf{O})} \otimes I_h^{(\mathbf{P})}) \\ \chi_{f;(g;h)}(a) &\triangleq \mathit{connect}(\chi_{g;h} \circ \pi_1, B)(\chi_f(a)) \text{ if } a \in \mathit{sup}(A^{*(\mathbf{O})} \otimes I_f^{(\mathbf{P})}) \\ \chi_{f;(g;h)}(a) &\triangleq \mathit{connect}(\chi_f \circ \pi_1^{-1}, B')(\chi_{g;h}(a)) \text{ if } a \in \mathit{sup}(D^{(\mathbf{O})} \otimes I_{g;h}^{(\mathbf{P})}) \end{aligned}$$

One way to see that these two bijective functions are equal is to view them as case trees, and consider every case. There are 13 such cases to consider, out of which three are not possible.

We show three cases:

1. If $a \in \mathit{sup}(A^{*(\mathbf{O})} \otimes I_f^{(\mathbf{P})})$, $\chi_f(a) \notin \mathit{sup}(B)$, and $\chi_f(a) \notin \mathit{sup}(C)$, then

$$\begin{aligned} &\chi_{(f;g);h}(a) \\ =&\mathit{connect}(\chi_h \circ \pi_2, C)(\chi_{f;g}(a)) \\ =&\mathit{connect}(\chi_h \circ \pi_2, C)(\chi_f(a)) \\ =&\chi_f(a) \end{aligned}$$

   and

$$\begin{aligned} &\chi_{f;(g;h)}(a) \\ =&\mathit{connect}(\chi_{g;h} \circ \pi_1, B)(\chi_f(a)) \\ =&\chi_f(a) \end{aligned}$$

   and thus equal.

2. Consider the case where $a \in \mathit{sup}(A^{*(\mathbf{O})} \otimes I_f^{(\mathbf{P})})$, $\chi_f(a) \notin \mathit{sup}(B)$, and $\chi_f(a) \in \mathit{sup}(C)$. This case is not possible, since $\mathit{sup}(C)$ is not a subset of the codomain of $\chi_f(a)$, which is $\mathit{sup}(A^{*(\mathbf{P})} \otimes B^{(\mathbf{P})} \otimes I_f^{(\mathbf{O})})$.

3. If $a \in sup(D^{(\mathbf{O})} \otimes I_h^{(\mathbf{P})})$, $\chi_h(a) \in sup(C')$, $\pi_2^{-1}(\chi_h(a)) \in sup(C^{(\mathbf{O})} \otimes I_g^{(\mathbf{P})})$, and $\chi_g(\pi_2^{-1}(\chi_h(a))) \in sup(B')$, then

$$\begin{aligned}&\chi_{(f;g);h}(a)\\ =&connect(\chi_{f;g} \circ \pi_2^{-1}, C')(\chi_h(a))\\ =&\chi_{f;g}(\pi_2^{-1}(\chi_h(a)))\\ =&connect(\chi_f \circ \pi_1^{-1}, B')(\chi_g(\pi_2^{-1}(\chi_h(a))))\\ =&\chi_f(\pi_1^{-1}(\chi_g(\pi_2^{-1}(\chi_h(a)))))\end{aligned}$$

and

$$\begin{aligned}&\chi_{f;(g;h)}(a)\\ =&connect(\chi_f \circ \pi_1^{-1}, B')(\chi_{g;h}(a))\\ =&connect(\chi_f \circ \pi_1^{-1}, B')(connect(\chi_g \circ \pi_2^{-1}, C')(\chi_h(a)))\\ =&connect(\chi_f \circ \pi_1^{-1}, B')(\chi_g(\pi_2^{-1}(\chi_h(a))))\\ =&\chi_f(\pi_1^{-1}(\chi_g(\pi_2^{-1}(\chi_h(a)))))\end{aligned}$$

and thus equal.

The other cases are done similarly.

- id_A is well-formed. For any interface A,

$$id_A \triangleq (\emptyset, \chi, A \Rightarrow A')$$

for an A' such that $\pi \vdash A =_{\mathbb{A}} A'$ and

$$\begin{aligned}\chi(a) &\triangleq \pi(a) \text{ if } a \in sup(A^{*(\mathbf{O})})\\ \chi(a) &\triangleq \pi^{-1}(a) \text{ if } a \in sup(A'^{(\mathbf{O})}.)\end{aligned}$$

according to the definition.

We need to show that χ is a bijection:

$$\begin{aligned}\chi &\in sup((A \Rightarrow A')^{(\mathbf{O})}) \to sup((A \Rightarrow A')^{(\mathbf{P})})\\ &= sup(A^{*(\mathbf{O})} \cup A'^{(\mathbf{O})}) \to sup(A^{*(\mathbf{P})} \cup A'^{(\mathbf{P})})\end{aligned}$$

This is true since π is a bijection in $sup(A) \to sup(A')$.

- id_A is an identity. For any morphism $f : A \to B$ we observe that $id_A; f$ is structurally equivalent to f, so by Theorem 2.1, $[\![id_A; f]\!] =_{\mathbb{A}} [\![f]\!]$.

The case for $f; id_B$ is similar.

□

We will now show that **HRAMnet** is a symmetric monoidal category:

- The tensor product of two objects A, B, $A \otimes B$ has already been defined. We define the tensor of two morphisms $f = (\overline{E}_f, \chi_f, A \Rightarrow B), g = (\overline{E}_g, \chi_g, C \Rightarrow D)$ as $f \otimes g = (\overline{E}_f \cup \overline{E}_g, \chi_f \otimes \chi_g, A \otimes C \Rightarrow B \otimes D)$.

- The unit object is the empty interface, $\emptyset$.
- Since $A \otimes (B \otimes C) = A \cup B \cup C = (A \otimes B) \otimes C$ we define the associator $\alpha_{A,B,C} \triangleq id_{A\otimes B\otimes C}$ with the obvious inverse.
- Similarly, since $\emptyset \otimes A = \emptyset \cup A = A = A \cup \emptyset = A \otimes \emptyset$, we define the left unitor $\lambda_A \triangleq id_A$ and the right unitor $\rho_A \triangleq id_A$.
- Since $A \otimes B = A \cup B = B \cup A = B \otimes A$ we define the commutativity constraint $\gamma_{A,B} \triangleq id_{A\otimes B}$.

Proposition 2.3. ***HRAMnet*** *is a symmetric monoidal category.*

Proof.
- The tensor product is well-defined, i.e. for two morphisms f, g, $f \otimes g$ is a well-formed net. This is easy to see since f and g are well-formed.
- The tensor product is a bifunctor:
  - $id_A \otimes id_B = (\emptyset, \chi_1 \otimes \chi_2, A \otimes B \Rightarrow A' \otimes B') = id_{A\otimes B}$ by the definition of $id_{A\otimes B}$.
  - $(f;g)\otimes(h;i) = f\otimes h; g\otimes i$ by the definition of composition and tensor on morphisms.
- The coherence conditions of the natural isomorphisms are trivial since the isomorphisms amount to identities.

□

Next we show that **HRAMnet** is a compact-closed category:

- We have already defined the dual A^* of an object A.
- Since $\emptyset \Rightarrow (A^* \otimes A') = \emptyset^* \cup (A^* \cup A') = A \Rightarrow A'$ we can define the unit $\eta_A \triangleq id_A$ and since $A \otimes A'^* \Rightarrow \emptyset = (A \cup A'^*)^* \cup \emptyset = A^* \cup A' = A \Rightarrow A'$ we can define the counit $\varepsilon_A \triangleq id_A$.

This leads us directly to the following result — what we set out to show:

Proposition 2.4. ***HRAMnet*** *is a symmetric compact-closed category.*

The following two theorems can be proved by induction on the trace length, and provide a connection between the **HRAMnet** tensor and composition and trace interleaving and composition.

Theorem 2.5. *If $f : A \to B$ and $g : C \to D$ are morphisms of* ***HRAMnet*** *then $[\![f \otimes g]\!] = [\![f]\!] \otimes [\![g]\!]$.*

Theorem 2.6. *If $f : A \to B$ and $g : B' \to C$ are morphisms of* ***HRAMnet*** *such that $\pi \vdash B =_{\mathbb{A}} B'$ then $[\![f;g]\!] = [\![f]\!];[\![g]\!]$.*

The following result explicates how communicating HRAMs can be combined into a single machine, where the intercommunication is done with jumping rather than message passing, in a sound way:

Theorem 2.7. *If $E_1 = (A_1, P_1)$ and $E_2 = (A_2, P_2)$ are engines and $S = (\{E_1, E_2\}, \chi, A)$ is a net, then $E_{12} = (A_1 \otimes A_2, P_1 \cup P_2)$ is an engine, $S' = (\{E_{12}\}, \chi, A)$ is a net and $[\![S]\!] \subseteq [\![S']\!]$.*

Proof. We show that for any trace s, $s \in [\![S]\!]$ implies $s \in [\![S']\!]$ by induction on the length of the trace.

Hypothesis. If $s \in [\![S]\!]$ and thus $\mathit{initial}(S) \xrightarrow{s} (\{(\overline{t}_1, h_1) : E_1, (\overline{t}_2, h_2) : E_2\}, \overline{m})$ for some sets of threads $\overline{t}_1$ and $\overline{t}_2$, heaps h_1 and h_2, and a multiset of messages $\overline{m}$, then $\mathit{initial}(S') \xrightarrow{s} (\{(\overline{t}_1 \cup \overline{t}_2 \cup \overline{t}_p, h_1 \cup h_2) : E_{12}\}, \overline{m}_p)$ where $\overline{t}_p$ is a set of threads and $\overline{m}_p$ is a multiset of messages such that:

1. each $t \in \overline{t}_p$ is on the form $t = (\texttt{spark}\ a, \overline{d})$ with $\chi(a) \in \mathit{sup}(A_1 \otimes A_2)$, and
2. $\overline{m} = \overline{m}_p \uplus \{(\chi(a), \mathit{msg}(\overline{d})) \mid (\texttt{spark}\ a, \overline{d}) \in \overline{t}_p\}$.

Intuitively, the net where E_1 and E_2 have been combined into one engine will not have pending messages (in $\overline{m}$) for communications between E_1 and E_2, but it can match the behaviour of such messages by threads that are just about to spark.

Base case. Since any net can take zero steps, the case when $s = \epsilon$ is trivial.

Inductive step. If $s = s'{::}\alpha$ and the hypothesis holds for s', then we have

$$\begin{aligned}
\mathit{initial}(S) &\xrightarrow{s'} (\{(\overline{t}_1, h_1) \ :\ E_1, (\overline{t}_2, h_2) \ :\ E_2\}, \overline{m}) \\
&\qquad \to^* \xrightarrow{\alpha} (\{(\overline{t}'_1, h'_1) \ :\ E_1, (\overline{t}'_2, h'_2) \ :\ E_2\}, \overline{m}') \\
\mathit{initial}(S') &\xrightarrow{s'} (\{(\overline{t}_1 \cup \overline{t}_2 \cup \overline{t}_p, h_1 \cup h_2) \ :\ E_{12}\}, \overline{m}_p)
\end{aligned}$$

with t_p and $\overline{m}'$ as in the hypothesis. We first show that S' can match the silent steps that S performs, by induction on the number of steps, using the same induction hypothesis as above:

Base case. Trivial.

Inductive step. Assume that we have

$$\begin{aligned}
\mathit{initial}(S) &\xrightarrow{s'}\to^* (\{(\overline{t}_1, h_1) \ :\ E_1, (\overline{t}_2, h_2) \ :\ E_2\}, \overline{m}) \\
\mathit{initial}(S') &\xrightarrow{s'}\to^* (\{(\overline{t}_1 \cup \overline{t}_2 \cup \overline{t}_p, h_1 \cup h_2) \ :\ E_{12}\}, \overline{m}_p)
\end{aligned}$$

Such that the induction hypothesis holds. We need to show that any step

$$\begin{gathered}
(\{(\overline{t}_1, h_1) \ :\ E_1, (\overline{t}_2, h_2) \ :\ E_2\}, \overline{m}) \to \\
(\{(\overline{t}'_1, h'_1) \ :\ E_1, (\overline{t}'_2, h'_2) \ :\ E_2\}, \overline{m}')
\end{gathered}$$

can be matched by (any number of) silent steps of the S' configuration, such that the induction hypothesis still holds.

- A thread of S performs a silent step. This is trivial, since the threads of the engine configuration of S' includes all threads of the configurations of S, and its heap is the union of those of S.

- A thread of S does an internal engine send step. Since $\overline{t}_1 \cup \overline{t}_2 \cup \overline{t}_p$ includes all threads of the S configuration, and for the port name a in question $\chi(a) \in A_1 \cup A_2 = A_1 \otimes A_2$, this can be matched by the configuration of S' such that the induction hypothesis still holds.
- A thread S does an external engine send. This means that there is a thread $t \in \overline{t}_1 \cup \overline{t}_2$ on the form $t = (\texttt{spark}\ a, \overline{d})$, which after the step will be removed, adding the message $(\chi(a), \mathit{msg}(\overline{d}))$ to its multiset of messages, i.e. $\overline{m}' = \overline{m} \uplus \{(\chi(a), \mathit{msg}(\overline{d}))\}$.
  If $\chi(a) \in A_1 \cup A_2$, then the configuration S' can take zero steps, and thus include t in the set of threads ready to spark. The induction hypothesis still holds, since $\overline{m}' = \overline{m} \uplus \{(\chi(a), \mathit{msg}(\overline{d}))\} = \overline{m}_p \uplus \{(\chi(a), \mathit{msg}(\overline{d})) \mid (\texttt{spark}\ a, \overline{d}) \in \overline{t}_p\} \uplus \{(\chi(a), \mathit{msg}(\overline{d}))\} = \overline{m}_p \uplus \{(\chi(a), \mathit{msg}(\overline{d})) \mid (\texttt{spark}\ a, \overline{d}) \in \overline{t}_p \cup \{t\}\}$.
  If $\chi(a) \in I$, then the configuration of S' can match the step of S, removing the thread t from also its set of threads. It is easy to see that the induction hypothesis holds also in this case.
- An engine of S receives a message. This means that $\overline{m} = \{(a, \overline{d})\} \uplus \overline{m}'$ for a message such that the port $(\mathbf{O}, a) \in A_1 \cup A_2 = A_1 \otimes A_2$. Then either $(a, \overline{d})$ is in $\overline{m}_p$ or in $\{(\chi(a), \mathit{msg}(\overline{d})) \mid (\texttt{spark}\ a, \overline{d}) \in \overline{t}_p\}$. If it is the former, E_{12} can receive the message and start a thread equal to that started in the configuration of S. If it is the latter, there is a thread $t = (\texttt{spark}\ \chi^{-1}(a), \overline{d}') \in \overline{t}_p$ with $\overline{d} = \mathit{msg}(\overline{d}')$ that can first take a send m step, adding it to the multiset of pending messages of the configuration of S', and then it can be received as in S.

Next we show that the α step can be matched: Assume that we have

$$\mathit{initial}(S) \xrightarrow{s'}\rightarrow^* (\{(\overline{t}_1, h_1) : E_1, (\overline{t}_2, h_2) : E_2\}, \overline{m})$$
$$\mathit{initial}(S') \xrightarrow{s'}\rightarrow^* (\{(\overline{t}_1 \cup \overline{t}_2 \cup \overline{t}_p, h_1 \cup h_2) : E_{12}\}, \overline{m}_p)$$

Such that the induction hypothesis holds. We need to show that for any α, a step

$$(\{(\overline{t}_1, h_1) : E_1, (\overline{t}_2, h_2) : E_2\}, \overline{m}) \xrightarrow{\alpha} (\{(\overline{t}_1', h_1') : E_1, (\overline{t}_2', h_2') : E_2\}, \overline{m}')$$

can be matched by the S' configuration, such that the induction hypothesis still holds. We have two cases:

- The configuration of S performs a send step. That is $\overline{m} = \{m\} \uplus \overline{m}'$ for an $m = (a, \overline{d})$ such that $(\mathbf{P}, a) \in A$. Since $\mathit{sup}(A)$ is disjoint from $\mathit{sup}(A_1 \cup A_2)$, the message is also in $\overline{m}_p$, so the configuration of S' can match the step.
- The configuration of S performs a receive step. This case is easy, as S and S' have the same interface A.

$\square$

We define a family of projection HRAM nets $\Pi_{i,A_1\otimes\cdots\otimes A_n} : A_1 \otimes \cdots \otimes A_n \to A_i$ by first constructing a family of "sinks" $!_A : A \to I \triangleq \mathit{singleton}((A \Rightarrow I, P))$ where $I = \emptyset$ and $P(a) = \mathtt{end}$ for each a in its domain and then defining e.g. $\Pi_{1,A\otimes B} : A \otimes B \to A \triangleq id_A \otimes !_B$.

3 Game nets for ICA

The structure of a **HRAMnet** token is determined by the number of registers r and the message size r_m, which are globally fixed. To implement game-semantic machines we require four message components: a port name, two pointer names, and a data fragment, meaning that $r_m = 3$. We choose $r = 4$, to get an additional register for temporary thread values to work with. From this point on, messages in nets and traces will be restricted to this form.

The message structure is intended to capture the structure of a move when game semantics is expressed in the nominal model. The port name is the move, the first name is the "point" whereas the second name is the "butt" of a justification arrow, and the data is the value of the move. This direct and abstract encoding of the justification pointer as names is quite different to that used in PAM and in other GOI-based token machines. In PAM the pointer is represented by a sequence of integers encoding the hereditary justification of the move, which is a snap-shot of the computational causal history of the move, just like in GOI-based machines. Such encodings have an immediate negative consequence, as tokens can become impractically large in complex computations, especially involving recursion. Large tokens entail not only significant communication overheads but also the computational overheads of decoding their structure. A subtler negative consequence of such an encoding is that it makes supporting the semantic structures required to interpret state and concurrency needlessly complicated and inefficient. The nominal representation is simple and compact, and efficiently exploits local machine memory (heap) in a way that previous abstract machines, of a "functional" nature, do not.

The price that we pay is a failure of compositionality, which we will illustrate shortly. The rest of the section will show how compositionality can be restored without substantially changing the HRAM framework. If in HRAM nets compositionality is "plug-and-play", as apparent from its compact-closed structure, *Game Abstract Machine* (GAM) composition must be mediated by a family of operators which are themselves HRAMs.

In this simple motivating example it is assumed that the reader is familiar with game semantics, and several of the notions to be introduced formally in the next sub-sections are anticipated. We trust that this will be not confusing.

Let S be a HRAM representing the game semantic model for the *successor* operation $S : \mathit{int} \to \mathit{int}$. The HRAM net in Fig. 4 represents a (failed) attempt to construct an interpretation for the term $x : \mathit{int} \vdash S(S(x)) : \mathit{int}$ in a context $C[-_{\mathit{int}}] : \mathit{int}$. This is the standard way of composing GOI-like machines.

The labels along the edges of the HRAM net trace a token (a, p_0, p_1, d) sent by the context $C[-]$ in order to evaluate the term. We elide a and d, which are irrelevant, to keep the diagram uncluttered. The token is received by S and

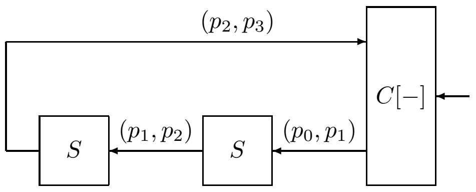

Figure 4: Non-locality of names in HRAM composition

propagated to the other S HRAM, this time with tokens (p_1, p_2). This trace of events $(p_0, p_1)::(p_1, p_2)$ corresponds to the existence of a justification pointer from the second action to the first in the game model. The essential correctness invariant for a well-formed trace representing a game-semantic play is that each token consists of a *known* name and a *fresh* name (if locally created, or *unknown* if externally created). However, the second S machine will respond with (p_2, p_3) to (p_1, p_2), leading to a situation where $C[-]$ receives a token formed from two unknown tokens.

In game semantics, the composition of $(p_0, p_1)::(p_1, p_2)$ with $(p_1, p_2)::(p_2, p_3)$ should lead to $(p_0, p_1)::(p_1, p_3)$, as justification pointers are "extended" so that they never point into a move hidden through composition. This is precisely what the composition operator, a specialised HRAM, will be designed to achieve.

3.1 Game abstract machines (GAM) and nets

Definition 3.1. *We define a* game interface *(cf.* arena*) as a tuple* $\mathfrak{A} = (A, \mathrm{qst}_{\mathfrak{A}}, \mathrm{ini}_{\mathfrak{A}}, \vdash_{\mathfrak{A}})$ *where*

- $A \in \mathcal{I}$ *is an interface. For game interfaces* $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}$ *we will write* A, B, C *and so on for their underlying interfaces.*
- *The set of ports is partitioned into a subset of question port names* $\mathrm{qst}_{\mathfrak{A}}$ *and one of answer port names* $\mathrm{ans}_{\mathfrak{A}}$, $\mathrm{qst}_{\mathfrak{A}} \uplus \mathrm{ans}_{\mathfrak{A}} = \sup(A)$.
- *The set of initial port names* $\mathrm{ini}_{\mathfrak{A}}$ *is a subset of the* **O***-labelled question ports.*
- *The enabling relation* $\vdash_{\mathfrak{A}}$ *relates question port names to non-initial port names such that if* $a \vdash_{\mathfrak{A}} a'$ *for port names* $a \in \mathrm{qst}_{\mathfrak{A}}$ *with* $(l, a) \in A$ *and* $a' \in \sup(A) \setminus \mathrm{ini}_{\mathfrak{A}}$ *with* $(l', a') \in A$*, then* $l \neq l'$.

For notational consistency, write $opp_{\mathfrak{A}} \triangleq sup(A^{(\mathbf{O})})$ and $prop_{\mathfrak{A}} \triangleq sup(A^{(\mathbf{P})})$. Call the set of all game interfaces $\mathcal{I}_{\mathfrak{G}}$. Game interfaces are equivariant, $\pi \vdash \mathfrak{A} =_{\mathbb{A}} \mathfrak{B}$, if and only if $\pi \vdash A =_{\mathbb{A}} B$, $\{\pi(a) \mid a \in qst_{\mathfrak{A}}\} = qst_{\mathfrak{B}}$, $\{\pi(a) \mid a \in ini_{\mathfrak{A}}\} = ini_{\mathfrak{B}}$ and $\{(\pi(a), \pi(a')) \mid a \vdash_{\mathfrak{A}} a'\} = \vdash_{\mathfrak{B}}$.

Definition 3.2. *For game interfaces (with disjoint sets of port names)* $\mathfrak{A}$ *and* $\mathfrak{B}$*, we define:*

$$\mathfrak{A} \otimes \mathfrak{B} \triangleq (A \otimes B, \mathrm{qst}_{\mathfrak{A}} \cup \mathrm{qst}_{\mathfrak{B}}, \mathrm{ini}_{\mathfrak{A}} \cup \mathrm{ini}_{\mathfrak{B}}, \vdash_{\mathfrak{A}} \cup \vdash_{\mathfrak{B}})$$

$$\mathfrak{A} \Rightarrow \mathfrak{B} \triangleq (A \Rightarrow B, \mathrm{qst}_{\mathfrak{A}} \cup \mathrm{qst}_{\mathfrak{B}}, \mathrm{ini}_{\mathfrak{B}}, \vdash_{\mathfrak{A}} \cup \vdash_{\mathfrak{B}} \cup (\mathrm{ini}_{\mathfrak{B}} \times \mathrm{ini}_{\mathfrak{A}})).$$

A *GAM net* is a tuple $G = (S, \mathfrak{A}) \in \mathcal{S} \times \mathcal{I}_{\mathfrak{G}}$ consisting of a net and a game interface such that $S = (\overline{E}, \chi, A)$, i.e. the interface of the game net is the same as that of the game interface. The denotational semantics of a GAM net $G = (S, \mathfrak{A})$ is just that of the underlying HRAM net: $[\![G]\!] \triangleq [\![S]\!]$.

3.2 Game traces

To be able to use game semantics as the specification for game nets we define the usual legality conditions on traces, following [9].

Definition 3.3. *The coabstracted and free pointers* cp *and* fp $\in$ traces $\to \mathcal{P}(\mathbb{P})$ *are:*

$$\begin{aligned}
\mathrm{cp}(\epsilon) &\triangleq \emptyset \\
\mathrm{cp}(s{::}(l,(a,p,p',d))) &\triangleq \mathrm{cp}(s) \cup \{p'\} \\
\mathrm{fp}(\epsilon) &\triangleq \emptyset \\
\mathrm{fp}(s{::}(l,(a,p,p',d)) &\triangleq \mathrm{fp}(s) \cup (\{p\} \setminus \mathrm{cp}(s))
\end{aligned}$$

The pointers of a trace $ptrs(s) = cp(s) \cup fp(s)$.

Definition 3.4. *Define* $\mathrm{enabled}_{\mathfrak{A}} \in \mathrm{traces}_A \to \mathcal{P}(\sup(A) \times \mathbb{P})$ *inductively as follows:*

$$\begin{aligned}
\mathrm{enabled}_{\mathfrak{A}}(\epsilon) &\triangleq \emptyset \\
\mathrm{enabled}_{\mathfrak{A}}(s{::}(l,(a,p,p',d))) &\triangleq \mathrm{enabled}_{\mathfrak{A}}(s) \cup \{(a',p') \mid a \vdash_{\mathfrak{A}} a'\}
\end{aligned}$$

Definition 3.5. *We define the following relations over traces:*

- *Write* $s' \leq s$ *if and only if there is a trace* s_1 *such that* $s'{::}s_1 = s$*, i.e.* s' *is a* prefix *of* s.
- *Write* $s' \leq s$ *if and only if there are traces* s_1, s_2 *such that* $s_1{::}s'{::}s_2 = s$*, i.e.* s' *is a* segment *of* s.

Definition 3.6. *For an arena* $\mathfrak{A}$ *and a trace* $s \in \mathrm{traces}_A$*, we define the following legality conditions:*

- s *has* unique pointers *when* $s'{::}(l,(a,p,p',d)) \leq s$ *implies* $p' \notin \mathrm{ptrs}(s')$.
- s *is* correctly labelled *when* $(l,(a,p,p',d)) \subseteq s$ *implies* $a \in \sup(A^{(l)})$.
- s *is* justified *when* $s'{::}(l,(a,p,p',d)) \leq s$ *and* $a \notin \mathrm{ini}_{\mathfrak{A}}$ *implies* $(a,p) \in \mathrm{enabled}_{\mathfrak{A}}(s')$.
- s *is* well-opened *when* $s'{::}(l,(a,p,p',d)) \leq s$ *implies* $a \in \mathrm{ini}_{\mathfrak{A}}$ *and* $s' = \epsilon$.
- s *is* strictly scoped *when* $(l,(a,p,p',d)){::}s' \subseteq s$ *with* $a \in \mathrm{ans}_{\mathfrak{A}}$ *implies* $p \notin \mathrm{fp}(s')$.

- s *is* strictly nested *when* $(l_1,(a_1,p,p',d_1))::s'::(l_2,(a_2,p',p'',d_2))::s''::(l_3,(a_3,p',p''',d_3)) \subseteq s$ *implies* $(l_4,(a_4,p'',-,d_4)) \subseteq s''$ *for port names* $a_1,a_2 \in \mathrm{qst}_{\mathfrak{A}}$ *and* $a_3,a_4 \in \mathrm{ans}_{\mathfrak{A}}$.

- s *is* alternating *when* $(l_1,m_1)::(l_2,m_2) \subseteq s$ *implies* $l_1 \neq l_2$.

Definition 3.7. *We say that a question message* $\alpha = (l,(a,p,p',d))$ *(*$a \in \mathrm{qst}_{\mathfrak{A}}$*) is* pending *in a trace* $s = s_1::\alpha::s_2$ *if and only if there is no answer* $\alpha' = (l',(a',p',p'',d')) \subseteq s_2$ *(*$a' \in \mathrm{ans}_{\mathfrak{A}}$*), i.e. the question has not been answered.*

Write $P_{\mathfrak{A}}$ for the subset of $traces_A$ consisting of the traces that have unique pointers, are correctly labelled, justified, strictly scoped and strictly nested.

For a set of traces P, write P^{alt} for the subset consisting of only alternating traces, and P^{st} (for single-threaded) for the subset consisting of only well-opened traces.

Definition 3.8. *If* $s \in \mathrm{traces}$ *and* $X \subseteq \mathbb{P}$*, define the* hereditarily justified *trace* $s \upharpoonright X$*, where inductively* $(s',X') = s \upharpoonright X$*:*

$$\begin{aligned}
\epsilon \upharpoonright X &\triangleq (\epsilon, X) \\
s::(l,(a,p,p',d)) \upharpoonright X &\triangleq (s'::(l,(a,p,p',d)), B \cup \{p'\}) && \textit{if } p \in X' \\
s::(l,(a,p,p',d)) \upharpoonright X &\triangleq (s', B) && \textit{if } p \notin X'
\end{aligned}$$

Write $s \upharpoonright X$ for s' when $s \upharpoonright X = (s',X')$ when it is convenient.

3.3 Copycat

The quintessential game-semantic behaviour is that of the *copy-cat strategy*, as it appears in various guises in the representation of all structural morphisms of any category of strategies. A copy-cat not only replicates the behaviour of its Opponent in terms of moves, but also in terms of justification structures. Because of this, the copy-cat strategy needs to be either history-sensitive (stateful) or the justification information needs to be carried along with the token. We take the former approach, in contrast to IAM and other GOI-inspired machines.

Consider the identity (or copycat) strategy on **com** $\Rightarrow$ **com**, where **com** is a two-move arena (one question, one answer). A typical play may look as in Fig. 5. The full lines represent justification pointers, and the trace (play) is represented nominally as

$$(r_4,p_0,p_1)::(r_2,p_1,p_2)::(r_1,p_2,p_3)::(r_3,p_1,p_4)::(d_3,p_4)\cdots$$

To preserve the justification structure, a copycat engine only needs to store "copycat links", which are shown as dashed lines in the diagram between question moves. In this instance, for an input on r_4, a heap value mapping a freshly created p_2 (the pointer to r_2) to p_1 (the pointer from r_4) is added.

The reason for mapping p_2 to p_1 becomes clear when the engine later gets an input on r_1 with pointers p_2 and p_3. It can then replicate the move to r_3, but

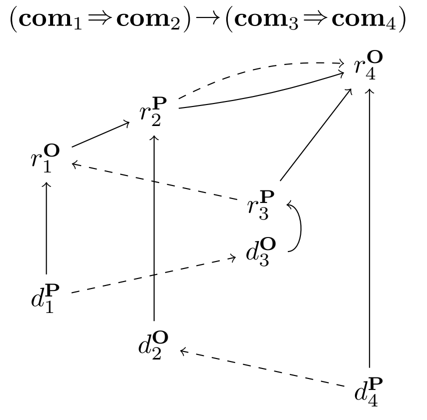

Figure 5: A typical play for copycat

using p_1 as a justifier. By following the p_2 pointer in the heap it gets p_1 so it can produce (r_3, p_1, p_4), where p_4 is a fresh heap value mapping to p_3. When receiving an answer, i.e. a d move, the copycat link can be dereferenced and then *discarded* from the heap.

The following HRAM macro-instructions are useful in defining copy-cat machines to, respectively, handle the pointers in an initial question, a non-initial question and an answer:

$$\begin{aligned}
\texttt{cci} &\triangleq \texttt{flip}\ 0, 1;\ 1 \leftarrow \texttt{new}\ 0, 3 \\
\texttt{ccq} &\triangleq 1 \leftarrow \texttt{new}\ 1, 3;\ 0, 3 \leftarrow \texttt{get}\ 0 \\
\texttt{cca} &\triangleq \texttt{flip}\ 0, 1;\ 0, 3 \leftarrow \texttt{get}1;\ \texttt{free}\ 1
\end{aligned}$$

For game interfaces $\mathfrak{A}$ and $\mathfrak{A}'$ such that $\pi \vdash \mathfrak{A} =_{\mathbb{A}} \mathfrak{A}'$, we define a generalised copycat engine as $\mathcal{CC}_{C,\pi,\mathfrak{A}} = (A \Rightarrow A', P)$, where:

$$\begin{aligned}
P \triangleq\ & \{q_2 \mapsto C;\ \texttt{spark}\, q_1 \mid q_2 \in \mathit{ini}_{\mathfrak{A}'} \wedge q_1 = \pi^{-1}(q_2)\} \\
& \cup \{q_2 \mapsto \texttt{ccq};\ \texttt{spark}\, q_1 \mid q_2 \in (\mathit{opp}_{\mathfrak{A}'} \cap \mathit{qst}_{\mathfrak{A}'}) \setminus \mathit{ini}_{\mathfrak{A}'} \wedge q_1 = \pi^{-1}(q_2)\} \\
& \cup \{a_2 \mapsto \texttt{cca};\ \texttt{spark}\, a_1 \mid a_2 \in \mathit{opp}_{\mathfrak{A}'} \cap \mathit{ans}_{\mathfrak{A}'} \wedge a_1 = \pi^{-1}(a_2)\} \\
& \cup \{q_1 \mapsto \texttt{ccq};\ \texttt{spark}\, q_2 \mid q_1 \in \mathit{opp}_{\mathfrak{A}} \cap \mathit{qst}_{\mathfrak{A}} \wedge q_2 = \pi(q_1)\} \\
& \cup \{a_1 \mapsto \texttt{cca};\ \texttt{spark}\, a_2 \mid a_1 \in \mathit{opp}_{\mathfrak{A}} \cap \mathit{ans}_{\mathfrak{A}} \wedge a_2 = \pi(a_1)\}
\end{aligned}$$

This copycat engine is parametrised with an initial instruction C, which is run when receiving an initial question. The engine for an ordinary copycat, i.e. the identity of games, is $\mathcal{CC}_{\texttt{cci},\pi,\mathfrak{A}}$. By slight abuse of notation, write $\mathcal{CC}_{\mathfrak{A}}$ for the singleton copycat game net $(\mathit{singleton}(\mathcal{CC}_{\texttt{cci},\pi,\mathfrak{A}}), \mathfrak{A} \Rightarrow \pi \cdot \mathfrak{A})$.

Following [9], we define a partial order $\leq$ over polarities, L, as $\mathbf{O} \leq \mathbf{O}, \mathbf{O} \leq \mathbf{P}, \mathbf{P} \leq \mathbf{P}$ and a preorder $\preccurlyeq$ over traces from $P_{\mathfrak{A}}$ to be the least reflexive and transitive such that if $l_1 \leq l_2$ then

$$\begin{aligned}
& s_1{::}(l_1, (a_1, p_1, p_1', d_1)){::}(l_2, (a_2, p_2, p_2', d_2)){::}s_2 \\
& \qquad\qquad \preccurlyeq s_1{::}(l_2, (a_2, p_2, p_2', d_2)){::}(l_1, (a_1, p_1, p_1', d_1)){::}s_2,
\end{aligned}$$

where $p'_1 \neq p_2$. A set of traces $S \subseteq P_{\mathfrak{A}}$ is *saturated* if and only if, for $s, s' \in P_{\mathfrak{A}}$, $s' \preccurlyeq s$ and $s \in S$ implies $s' \in S$. If $S \subseteq P_{\mathfrak{A}}$ is a set of traces, let $sat(S)$ be the smallest saturated set of traces that contains S.

The usual definition of the copycat strategy (in the alternating and single-threaded setting) as a set of traces is

$$\text{æ}^{st,alt}_{\mathfrak{A},\mathfrak{A}'} \triangleq \{s \in P^{st,alt}_{\mathfrak{A}\Rightarrow\mathfrak{A}'} \mid \forall s' \leq_{\text{even}} s.\, s'^{*} \upharpoonright A =_{\mathbb{AP}} s' \upharpoonright A'\}$$

Definition 3.9. *A set of traces S_1 is* **P**-closed *with respect to a set of traces S_2 if and only if $s' \in S_1 \cap S_2$ and $s = s'{::}(\mathbf{P}, (a, p, p', d)) \in S_1$ implies $s \in S_2$.*

The intuition of **P**-closure is that if the trace s' is "legal" according to S_2, then any outputs that can occur after s' in S_1 are also legal.

Definition 3.10. *We say that a GAM net f* implements *a set of traces S if and only if $S \subseteq [\![f]\!]$ and $[\![f]\!]$ is* **P***-closed with respect to S.*

This is the form of the statements of correctness for game nets that we want; it certifies that the net f can accommodate all traces in S and, furthermore, that it only produces legal outputs when given valid inputs.

The main result of this section establishes the correctness of the GAM for copycat.

Theorem 3.11. *$\text{Æ}_{\pi,\mathfrak{A}}$ implements $\text{æ}_{\mathfrak{A},\pi\cdot\mathfrak{A}}$.*

This is a direct corollary of the Lem. 3.13,3.16,3.17, 3.18, and 3.22 given below.

Lemma 3.12. *If $n_1 = (\overline{e\,:\,E}, \overline{m})$ and $n'_1 = (\overline{e'\,:\,E}, \overline{m'})$ are net configurations of a net $f = (\overline{E}, \chi, A)$, and $n_1 \xrightarrow{(x)} n'_1$ ($(x) \in \{\bullet\} \cup (L \times \mathcal{M}_{\sup(A)})$) then $n_2 \xrightarrow{(x)} n'_2$ where $n_2 = (\overline{e\,:\,E}, \overline{m} \uplus \{m\})$ and $n'_2 = (\overline{e'\,:\,E}, \overline{m'} \uplus \{m\})$.*

Proof. By cases on (x):

- If $(x) = \bullet$, then $\overline{e\,:\,E} = \{e\ :\ E\} \cup \overline{e_0\,:\,E_0}$, $e \xrightarrow[E,\chi]{(y)} e'$ for some (y), $\overline{e'\,:\,E} = \{e'\ :\ E\} \cup \overline{e'_0\,:\,E_0}$. We have three cases for (y):
  - If $(y) = \bullet$, then $e \xrightarrow[E,\chi]{} e'$ and $\overline{m'} = \overline{m}$. Then we also have $n_2 = (\{e\ :\ E\} \cup \overline{e_0\,:\,E_0}, \overline{m} \uplus \{m\}) \to (\{e'\ :\ E\} \cup \overline{e'_0\,:\,E_0}, \overline{m} \uplus \{m\}) = n'_2$.
  - If $(y) = (\mathbf{P}, m')$, then $e \xrightarrow[E,\chi]{m'} e'$ and $\overline{m'} = \{m'\} \cup \overline{m}$. Then we also have $n_2 = (\{e : E\} \cup \overline{e_0\,:\,E_0}, \overline{m} \uplus \{m\}) \to (\{e' : E\} \cup \overline{e'_0\,:\,E_0}, \{m'\} \uplus \overline{m} \uplus \{m\}) = n'_2$.
  - If $(y) = (\mathbf{O}, m')$, then $e \xrightarrow[E,\chi]{\overline{m'}} e'$ and $\overline{m} = \{m'\} \uplus \overline{m'}$. Then we also have $n_2 = (\{e\ :\ E\} \cup \overline{e_0\,:\,E_0}, \{m'\} \uplus \overline{m'} \uplus \{m\}) \to (\{e'\ :\ E\} \cup \overline{e'_0\,:\,E_0}, \overline{m'} \uplus \{m\}) = n'_2$.
- If $(x) = (\mathbf{P}, m')$, then $\overline{e'\,:\,E} = \overline{e\,:\,E}$ and $\overline{m} = \{m'\} \uplus \overline{m'}$. Then we also have $n_2 = (\overline{e\,:\,E}, \{m'\} \uplus \overline{m'} \uplus \{m\}) \xrightarrow{m'} (\overline{e\,:\,E}, \overline{m'} \uplus \{m\}) = n'_2$.

- If $(x) = (\mathbf{O}, m')$, where $m' = (a, p, p', d)$ then $\overline{e' : E} = \overline{e : E}$ and $\overline{m}' = \{(\chi(a), p, p', d)\} \uplus \overline{m}$. Then we also have $n_2 = (\overline{e : E}, \overline{m} \uplus \{m\}) \xrightarrow{\overline{m'}} (\overline{e : E}, \{(\chi(a), p, p', d)\} \uplus \overline{m} \uplus \{m\}) = n_2'$.

□

Lemma 3.13. *If f is a net and s a trace, then*

1. $s = s_1{::}(l, m_1){::}(\mathbf{O}, m){::}s_2 \in [\![f]\!]$ *with witness* $\text{initial}(f) \xrightarrow{s} n$ *implies* $s' = s_1{::}(\mathbf{O}, m){::}(l, m_1){::}s_2 \in [\![f]\!]$ *with* $\text{initial}(f) \xrightarrow{s'} n$ *and*
2. $s = s_1{::}(\mathbf{P}, m){::}(l, m_1){::}s_2 \in [\![f]\!]$ *with witness* $\text{initial}(f) \xrightarrow{s} n$ *implies* $s' = s_1{::}(l, m_1){::}(\mathbf{P}, m){::}s_2 \in [\![f]\!]$ *with* $\text{initial}(f) \xrightarrow{s'} n$.

A special case of this theorem is that if $G = (f, \mathfrak{A})$ and, for a set of traces $S \subseteq P_{\mathfrak{A}}$, $S \subseteq [\![G]\!]$ holds, then $sat(S) \subseteq [\![G]\!]$.

Proof. 1. $s = s_1{::}(l, m_1){::}(\mathbf{O}, m){::}s_2 \in [\![f]\!]$ means that

$$initial(f) \xrightarrow{s_1}\xrightarrow{(x)}{}^{*}\ n_1 \xrightarrow{(l,m_1)} n_2 \xrightarrow{(y)}{}^{*} \xrightarrow{(\mathbf{O},m)} n_3 \xrightarrow{(z)}{}^{*} \xrightarrow{s_2} n_4$$

for net configurations n_1, n_2, n_3, n_4. For clarity, we take $(x), (y), (z)$ to be "names" for the silent transitions. We show that there exist n_2' and (y') such that

$$initial(f) \xrightarrow{s_1}\xrightarrow{(x)}{}^{*}\ n_1 \xrightarrow{(\mathbf{O},m)}\xrightarrow{(l,m_1)} n_2' \xrightarrow{(y')}{}^{*}\ n_3 \xrightarrow{(z)}\xrightarrow{s_2} n$$

by induction on the length of $\xrightarrow{(y)}{}^{*}$:

- Base case. If $\xrightarrow{(y)}{}^{*}$ is the identity relation, then assume

$$n_1 \xrightarrow{(l,m_1)} n_2 \xrightarrow{(\mathbf{O},m)} n_3$$

  Let $n_1 = (\overline{e_1 : E}, \overline{m}_1)$, $n_2 = (\overline{e_2 : E}, \overline{m}_2)$, $m = (a, p, p', d)$, and $m' = (\chi(a), p, p', d)$. Then $n_3 = (\overline{e_2 : E}, \{m'\} \uplus \overline{m}_2)$ by the definition of $\rightarrow$. Since $(\mathbf{O}, a) \in I$, $n_1 \xrightarrow{(\mathbf{O},m)} (\overline{e_1 : E}, \{m'\} \uplus \overline{m}_1)$. Also, since $n_1 \xrightarrow{(l,m_1)} n_2$ we have $(\overline{e_1 : E}, \{m'\} \uplus \overline{m}_2) \xrightarrow{(l,m_1)} n_3$ by Lemma 3.12. Composing the relations, we get

$$n_1 \xrightarrow{(\mathbf{O},m)}\xrightarrow{(l,m_1)} n_3$$

  which completes the base case.

- Inductive step. If $\xrightarrow{(y)}{}^{*} = \xrightarrow{(y_0)}{}^{*} \xrightarrow{\bullet}$ such that for any n_3'

$$n_1 \xrightarrow{(l,m_1)} n_2 \xrightarrow{(y_0)}{}^{*} \xrightarrow{(\mathbf{O},m)} n_3'$$

  implies that there exist n_2' and (y_0') with

$$n_1 \xrightarrow{(\mathbf{O},m)}\xrightarrow{(l,m_1)} n_2' \xrightarrow{(y_0')}{}^{*}\ n_3'$$

then assume

$$n_1 \xrightarrow{(l,m_1)} n_2 \xrightarrow{(y_0)}{}^{*} n_{y_0} \overset{\bullet}{\rightarrow} n_y \xrightarrow{(\mathbf{O},m)} n_3$$

Let $n_{y_0} = (\overline{e_{y_0} : E}, \overline{m}_{y_0})$, $n_y = (\overline{e_y : E}, \overline{m}_y)$, $m = (a, p, p', d)$, and $m' = (\chi(a), p, p', d)$. Then $n_3 = (\overline{e_y : E}, \{m'\} \uplus \overline{m}_y)$ by the definition of $\rightarrow$. Since $(\mathbf{O}, a) \in I$, $n_{y_0} \xrightarrow{(\mathbf{O},m)} (\overline{e_{y_0} : E}, \{m'\} \uplus \overline{m}_{y_0})$. Also, since $n_{y_0} \overset{\bullet}{\rightarrow} n_y$ we have $(\overline{e_{y_0} : E}, \{m'\} \uplus \overline{m}_{y_0}) \overset{\bullet}{\rightarrow} n_3$ by Lemma 3.12. Composing the relations, we get

$$n_1 \xrightarrow{(l,m_1)} n_2 \xrightarrow{(y_0)}{}^{*} n_{y_0} \xrightarrow{(\mathbf{O},m)} (\overline{e_{y_0} : E}, \{m'\} \uplus \overline{m}_{y_0}) \overset{\bullet}{\rightarrow} n_3$$

Applying the hypothesis, we finally get

$$n_1 \xrightarrow{(\mathbf{O},m)}\xrightarrow{(l,m_1)} n_2' \xrightarrow{(y_0')}{}^{*} \overset{\bullet}{\rightarrow} n_3$$

which completes the first part of the proof.

2. $s = s_1{::}(\mathbf{P}, m){::}(l, m_1){::}s_2 \in [\![f]\!]$ means that

$$\mathit{initial}(f) \xrightarrow{s_1}\xrightarrow{(x)}{}^{*} n_1 \xrightarrow{(\mathbf{P},m)} n_2 \xrightarrow{(y)}{}^{*} \xrightarrow{(l,m_1)} n_3 \xrightarrow{(z)}{}^{*} \xrightarrow{s_2} n_4$$

for net configurations n_1, n_2, n_3, n_4 and $(x), (y), (z)$ names for the silent transitions. We show that there exist (y') and n_2' such that

$$\mathit{initial}(f) \xrightarrow{s_1}\xrightarrow{(x)}{}^{*} n_1 \xrightarrow{(y')}{}^{*} n_2' \xrightarrow{(l,m_1)}\xrightarrow{(\mathbf{P},m)} n_3 \xrightarrow{(z)}{}^{*} \xrightarrow{s_2} n$$

by induction on the length of $\xrightarrow{(y)}{}^{*}$:

- Base case. If $\xrightarrow{(y)}{}^{*}$ is the identity relation, then assume

$$n_1 \xrightarrow{(\mathbf{P},m)} n_2 \xrightarrow{(l,m_1)} n_3$$

Let $n_2 = (\overline{e_2 : E}, \overline{m}_2)$, $n_3 = (\overline{e_3 : E}, \overline{m}_3)$, $m = (a, p, p', d)$ Then $n_1 = (\overline{e_2 : E}, \{m\} \uplus \overline{m}_2)$ by the definition of $\rightarrow$. Since $(\mathbf{P}, a) \in I$, $(\overline{e_3 : E}, \{m\} \uplus \overline{m}_3) \xrightarrow{(\mathbf{P},m)} n_3$. Also, since $n_2 \xrightarrow{(l,m_1)} n_3$ we have $n_1 \xrightarrow{(l,m_1)} (\overline{e_3 : E}, \{m\} \uplus \overline{m}_3)$ by Lemma 3.12. Composing the relations, we get

$$n_1 \xrightarrow{(l,m_1)}\xrightarrow{(\mathbf{P},m)} n_3$$

which completes the base case.

- Inductive step. If $\xrightarrow{(y)}{}^{*} = \overset{\bullet}{\rightarrow}\xrightarrow{(y_0)}{}^{*}$ such that for any n_1'

$$n_1' \xrightarrow{(\mathbf{P},m)}\xrightarrow{(y_0)}{}^{*} n_2 \xrightarrow{(l,m_1)} n_3$$

implies that there exist n_2' and (y_0') with

$$n_1' \xrightarrow{(y_0')}{}^{*} n_2' \xrightarrow{(l,m_1)}\xrightarrow{(\mathbf{P},m)} n_3$$

then assume

$$n_1 \xrightarrow{(\mathbf{P},m)} n_m \overset{\bullet}{\rightarrow} n_y \xrightarrow{(y_0)}{}^{*} n_2 \xrightarrow{(l,m_1)} n_3$$

Let $n_m = (\overline{e_m : E}, \overline{m}_m)$, $n_y = (\overline{e_y : E}, \overline{m}_y)$, and $m = (a, p, p', d)$. Then $n_1 = (\overline{e_m : E}, \{m\} \uplus \overline{m}_m)$ by the definition of $\rightarrow$. Since $(\mathbf{P}, a) \in I$, $(\overline{e_y : E}, \{m\} \uplus \overline{m}_y) \xrightarrow{(\mathbf{P},m)} n_y$. Also, since $n_m \overset{\bullet}{\rightarrow} n_y$ we have $n_1 \overset{\bullet}{\rightarrow} (\overline{e_y : E}, \{m\} \uplus \overline{m}_y)$ by Lemma 3.12. Composing the relations, we get

$$n_1 \overset{\bullet}{\rightarrow} (\overline{e_y : E}, \{m\} \uplus \overline{m}_y) \xrightarrow{(\mathbf{P},m)} n_y \xrightarrow{(y_0)}{}^{*} n_2 \xrightarrow{(l,m_1)} n_3$$

Applying the hypothesis, we finally get

$$n_1 \overset{\bullet}{\rightarrow} \xrightarrow{(y_0')}{}^{*} n_2' \xrightarrow{(l,m_1)} \xrightarrow{(\mathbf{P},m)} n_3$$

which completes the proof.

$\square$

Lemma 3.14. *If $s, s' \in P_{\mathfrak{A}}$ and $s' \preccurlyeq s$, then*

1. enabled(s) = enabled(s'),
2. cp(s) = cp(s'), *and*
3. fp(s) = fp(s').

Proof. Induction on $\preccurlyeq$. The base case is trivial. Consider the case where $s = s_1{::}\alpha_2{::}\alpha_1{::}s_2$ and $s' = s_1{::}\alpha_1{::}\alpha_2{::}s_2$. Let $\alpha_1 = (l, (a_1, p_1, p_1', d_1))$ and $\alpha_2 = (l, (a_2, p_2, p_2', d_2))$.

1. Induction on the length of s_2. In the base case, we have (by associativity of $\cup$): $\mathit{enabled}(s_1{::}\alpha_2{::}\alpha_1) = \mathit{enabled}(s_1) \cup \{(a, p_2') \mid a_2 \vdash_{\mathfrak{A}} a\} \cup \{(a, p_1') \mid a_1 \vdash_{\mathfrak{A}} a\} = \mathit{enabled}(s_1) \cup \{(a, p_1') \mid a_1 \vdash_{\mathfrak{A}} a\} \cup \{(a, p_2') \mid a_2 \vdash_{\mathfrak{A}} a\}$.
2. Induction on the length of s_2 as in 1.
3. Induction on the length of s_2. In the base case, we have (since by the def. of $\preccurlyeq$, $p_1 \neq p_2'$ and $p_2 \neq p_1'$):

$$\begin{aligned}
\mathit{fp}(s_1{::}\alpha_2{::}\alpha_1) &= \\
\mathit{fp}(s_1{::}\alpha_2) \cup (\{p_1\} \setminus \mathit{cp}(s_1{::}\alpha_2)) &= \\
\mathit{fp}(s_1) \cup (\{p_2\} \setminus \mathit{cp}(s_1)) \cup (\{p_1\} \setminus (\mathit{cp}(s_1) \cup \{p_2'\})) &= \\
\mathit{fp}(s_1) \cup (\{p_2\} \setminus (\mathit{cp}(s_1) \cup \{p_1'\})) \cup (\{p_1\} \setminus \mathit{cp}(s_1)) &= \\
\mathit{fp}(s_1) \cup (\{p_1\} \setminus \mathit{cp}(s_1)) \cup (\{p_2\} \setminus (\mathit{cp}(s_1) \cup \{p_1'\})) &= \\
\mathit{fp}(s_1{::}\alpha_1) \cup (\{p_2\} \setminus \mathit{cp}(s_1{::}\alpha_1)) &= \\
\mathit{fp}(s_1{::}\alpha_1{::}\alpha_2)
\end{aligned}$$

$\square$

Lemma 3.15. *Let $S \subseteq P_{\mathfrak{A}}$ be a saturated set of traces. If $s, s' \in S$ are traces such that $s' \preccurlyeq s$ and $s{::}\alpha \in S$, then $s'{::}\alpha \in S$.*

Proof. Induction on $\preccurlyeq$. The base case is trivial. We show the case of a single swapping. If $s' \preccurlyeq s$, we have $s = s_1{::}\alpha_2{::}\alpha_1{::}s_2$ and $s' = s_1{::}\alpha_1{::}\alpha_2{::}s_2$ for some $s_1, s_2, \alpha_1, \alpha_2$. Obviously, $s'{::}\alpha \preccurlyeq s{::}\alpha$.

We have to show that if $s{::}\alpha \in P_{\mathfrak{A}}$, then $s'{::}\alpha \in P_{\mathfrak{A}}$. We have to show that $s'{::}\alpha$ fulfils the legality conditions imposed by $P_{\mathfrak{A}}$:

- It is easy to see that $s'{::}\alpha$ has unique pointers and is correctly labelled.
- $s'{::}\alpha$ is justified since $\mathit{enabled}(s) = \mathit{enabled}(s')$ by Lemma 3.14.
- To see that $s'{::}\alpha$ strictly scoped, consider the ("worst") case when

$$(l, (a, p, p', d)){::}s_3{::}\alpha \subseteq s'{::}\alpha \text{ and } a \in \mathit{ans}_{\mathfrak{A}}$$

  (i.e. we pick the segment that goes right up to the end of the trace). We consider the different possibilities of the position of this answer message:

  - If $(l, (a, p, p', d)) \subseteq s_1$, then let $s'_4 = (l, (a, p, p', d)){::}s'_1{::}\alpha_1{::}\alpha_2{::}s_2{::}\alpha \subseteq s'{::}\alpha$ and $s_4 = (l, (a, p, p', d)){::}s'_1{::}\alpha_2{::}\alpha_1{::}s_2{::}\alpha$. We also know that $p \notin \mathit{fp}(s_4)$ as $s{::}\alpha \in P_{\mathfrak{A}}$. Now, since $s'_4 \preccurlyeq s_4$, we have $\mathit{fp}(s_4) = \mathit{fp}(s'_4)$ by Lemma 3.14 and thus also $p \notin \mathit{fp}(s'_4)$.
  - If $(l, (a, p, p', d)) = \alpha_2$. We know that $p \notin \mathit{fp}(s_2{::}\alpha)$ by $s{::}\alpha \in P_{\mathfrak{A}}$. Since $s' \in P_{\mathfrak{A}}$ we have $p \notin \mathit{fp}(\alpha_1)$ and can so conclude that $p \notin \mathit{fp}(\alpha_1{::}s_2{::}\alpha)$.
  - If $(l, (a, p, p', d)) = \alpha_1$ or $(l, (a, p, p', d)) \subseteq s_2$, $p \notin \mathit{fp}(s_2{::}\alpha)$ follows immediately from $s \in P_{\mathfrak{A}}$.
  - If $(l, (a, p, p', d)) = \alpha$, $p \notin \mathit{fp}(\epsilon) = \emptyset$ is trivially true.

- To see that $s'{::}\alpha$ is strictly nested, assume

$$(l_1, (a_1, p, p', d_1)){::}s_1{::}(l_2, (a_2, p', p'', d_2)){::}s_2{::}(l_3, (a_3, p', p''', d_3)) \subseteq s'{::}\alpha$$

  for port names $a_1, a_2 \in \mathit{qst}_{\mathfrak{A}}$ and $a_3 \in \mathit{ans}_{\mathfrak{A}}$. We have to show that this implies $(l_4, (a_4, p'', -, d_4)) \subseteq s_2$, for a port name $a_4 \in \mathit{ans}_{\mathfrak{A}}$. We proceed by considering the possible positions of the last message in the segment:

  - If $(l_3, (a_3, p', p''', d_3)) \subseteq s'$, then the proof is immediate, by $s' \in P_{\mathfrak{A}}$ being strictly nested.
  - If $(l_3, (a_3, p', p''', d_3)) = \alpha$ we use the fact that $s{::}\alpha \in P_{\mathfrak{A}}$ is strictly nested. We assume that the implication (using the same names) as above holds but instead for $s{::}\alpha$, and show that any swappings that can have occurred in s' that reorder the a_1, a_2, a_4 moves would render s' illegal:
    - If a_2 was moved before a_1, then s' would not be justified.
    - If a_4 was moved before a_2, then s' would not be justified.

    As the order is preserved, this shows that the swappings must be done in a way such that the implication holds for $s'{::}\alpha$.

$\square$

Lemma 3.16. *For any game net $f = (S, \mathfrak{A})$ and trace $s \in P_{\mathfrak{A}}$, $s \in [\![f]\!]$ if and only if $\forall p \in \mathrm{fp}(s).s \upharpoonright \{p\} \in [\![f]\!]$.*

Lemma 3.17. $\text{æ}^{\text{st,alt}}_{\mathfrak{A},\pi\cdot\mathfrak{A}} \subseteq [\![\text{Œ}_{\pi,\mathfrak{A}}]\!]$.

Proof. For convenience, let $(f, \mathfrak{A} \Rightarrow \mathfrak{A}') = \text{Œ}_{\pi,\mathfrak{A},\mathfrak{A}'}$, $S_1 = \text{æ}^{st,alt}_{\mathfrak{A},\mathfrak{A}'}$ and $S_2 = [\![f]\!]$. We show that $s \in S_1$ implies $s \in S_2$, by induction on the length of s:

- Hypothesis. If s has even length, then $\mathit{initial}(f) \xrightarrow{s} (\{(\emptyset, h) : E\}, \emptyset)$ and h is exactly (nothing more than) a copycat heap for s over $\mathfrak{A} \Rightarrow \mathfrak{A}'$. In other words, there are no threads running and no pending messages and the heap is precisely specified.
- Base case. Trivial.
- Inductive step. At any point in the execution of the configuration of f, an **O**-labelled message can be received, so that case is rather uninteresting. Since the trace s is alternating, we consider two messages in each step:

  Assume $s = s'$::$(\mathbf{O}, (a_1, p_1, p'_1, d_1))$::$(\mathbf{P}, (a_2, p_2, p'_2, d_2)) \in S_1$ and that $s' \in S_2$. From the definition of æ we know that $a_2 = \tilde{\pi}_{\mathbb{A}}(a_1)$, $p_2 = \tilde{\pi}_{\mathbb{P}}(p_1)$, $p'_2 = \tilde{\pi}_{\mathbb{P}}(p_2)$, and $d_1 = d_2$.

  We are given that $\mathit{initial}(f) \xrightarrow{s}{}' (\{(\emptyset, h) : E\}, \emptyset)$ as in the hypothesis. We have five cases for the port name a_1. We show the first three, as the others are similar. In each case our single engine will receive a message and start a thread:

  - If $a_1 \in \mathit{ini}_{\mathfrak{A}'}$, then (since s is justified) $p_2 = p'_1$ and (by the definition of $\pi'_{\mathbb{A}}$) $a_2 = \pi_{\mathbb{A}}^{-1}(a_1)$. The engine runs the first clause of the copycat definition, and chooses to create the pointer p_2 and then performs a send operation. We thus get:

    $$\mathit{initial}(f) \xrightarrow{s} (\{(\emptyset, h \cup \{p'_2 \mapsto p'_1\})\}, \emptyset)$$

    It can easily be verified that the hypothesis holds for this new state.
  - If $a_1 \in (\mathit{opp}_{\mathfrak{A}'} \cap \mathit{qst}_{\mathfrak{A}'}) \setminus \mathit{ini}_{\mathfrak{A}'}$, then $a_2 = \pi_{\mathbb{A}}^{-1}(a_1)$. Since s is justified and strictly nested, there is a message $(\mathbf{P}, (a_3, p_3, p_1, d_3)) \subseteq s'$ that is pending.

    By the hypothesis there is a message $(\mathbf{O}, (\pi'_{\mathbb{A}}(a_3), p_4, p'_4, d_4)) \subseteq s'$ with $h(p_1) = p'_4$, which means that the `ccq` instruction can be run, yielding the following:

    $$\mathit{initial}(f) \xrightarrow{s} (\{(\emptyset, h \cup \{p'_2 \mapsto p'_1\})\}, \emptyset)$$

    The hypothesis can easily be verified also in this new state.
  - If $a_1 \in \mathit{opp}_{\mathfrak{A}'} \cap \mathit{ans}_{\mathfrak{A}'}$, then $a_2 = \pi_{\mathbb{A}}^{-1}(a_1)$. Since s is justified and strictly nested, there is a prefix s_1::$(\mathbf{P}, (a_3, p_3, p_1, d_3)) \leq s'$ whose last message is a pending question. By the hypothesis s_1 is then on the form $s_1 = s_2$::$(\mathbf{O}, (\pi'_{\mathbb{A}}(a_3), \tilde{\pi}_{\mathbb{P}}(p_3), \tilde{\pi}_{\mathbb{P}}(p_1), d_4))$ with $h = h' \cup \{p_1 \mapsto$

$\tilde{\pi}_{\mathbb{P}}(p_1)\}$, which means that the `cca` instruction can be run, yielding the following:

$$initial(f) \xrightarrow{s} (\{(\emptyset, h')\}, \emptyset)$$

The hypothesis is still true; the a_3 question is no longer pending and its pointer is removed from the heap (notice that $p_2 = \tilde{\pi}_{\mathbb{P}}(p_1)$).

□

Theorem 3.18. *If $s = s_1{::}o{::}s_2 \in \text{æ}_{\mathfrak{A},\mathfrak{A}'}$ and $p \not\subseteq s_2$, then $s{::}p \in \text{æ}_{\mathfrak{A},\mathfrak{A}'}$, where $o = (\mathbf{O}, (a, p, p', d))$ and $p = (\mathbf{P}, (\tilde{\pi}_{\mathbb{A}}(a), \tilde{\pi}_{\mathbb{P}}(p), \tilde{\pi}_{\mathbb{P}}(p'), d))$ (i.e. the “copy” of o).*

Proof. By induction on $\preccurlyeq$.

- Base case. This means that $s = s_1{::}o{::}s_2 \in \text{æ}^{alt}_{\mathfrak{A},\mathfrak{A}'}$. But since $p \not\subseteq s_2$ and by the definition of the alternating copycat, $s_2 = \epsilon$. It is easy to check that $s{::}p \in \text{æ}^{alt}_{\mathfrak{A},\mathfrak{A}'}$ and that it is legal.
- Inductive step. Assume $s \preccurlyeq s'$ for an $s' \in P_{\mathfrak{A}\Rightarrow\mathfrak{A}'}$ such that $s'{::}p \in \text{æ}_{\mathfrak{A},\mathfrak{A}'}$. By Lemma 3.15, $s{::}p \in \text{æ}_{\mathfrak{A},\mathfrak{A}'}$.

□

Definition 3.19. *Define the multiset of messages that a net configuration n is ready to immediately send as* $\text{ready}(n) \triangleq \{(\mathbf{P}, m) \mid \exists n'.\, n \rightarrow^* \xrightarrow{(\mathbf{P},m)} n'\}$.

Definition 3.20. *If s is a trace, h is a heap, $\mathfrak{A}$ is a game interface, and $\pi_{\mathbb{P}}$ is a permutation over $\mathbb{P}$, we say that h is a* copycat heap *for s over $\mathfrak{A}$ if and only if:*

For every pending $\mathbf{P}$-question from $\mathfrak{A}$ in s, i.e. $(\mathbf{P}, (a, p, p', d)) \subseteq s$ ($a \in \text{qst}_{\mathfrak{A}}$), $h(p') = (\tilde{\pi}_{\mathbb{P}}(p'), \emptyset)$.

Lemma 3.21. *If $s \in \text{æ}$ is a trace such that* $\text{initial}(\text{Æ}) \xrightarrow{s} n$*, then the following holds:*

1. *If $n \rightarrow^* n'$ then* $\text{ready}(n) = \text{ready}(n')$.
2. *If $n \rightarrow^* \xrightarrow{(\mathbf{P},m)} n'$, then* $\text{ready}(n) = \text{ready}(n') \cup \{(\mathbf{P}, m)\}$.

As we are only interested in what is observable, the trace s is thus equivalent to one where silent steps are only taken in one go by one thread right before outputs.

Proof. 1. For convenience, we give the composition of silent steps a name, $n \xrightarrow{(x)}{}^{*} n'$. We proceed by induction on the length of (x):

- Base case. Immediate.
- Inductive step. If $n \rightarrow \xrightarrow{(x')}{}^{*} n'$, we analyse the first silent step, which means that a thread t of the engine in the net takes a step:

  - In the cases where an instruction that does not change or depend on the heap is run, the step cannot affect *ready*(n).
  - In the case where the instruction is in $\{\texttt{cci}, \texttt{ccq}, \texttt{exi}, \texttt{exq}\}$, we note that the heap is not *changed*, but merely extended with a fresh mapping which can not have appeared earlier in the trace.
  - If the instruction is `cca`, since the trace s is strictly nested by assumption, the input message that this message stems from occurs in a position in the trace where it would later be illegal to mention the deallocated pointer again.

2. Immediate.

□

Theorem 3.22. *If $s \in \mathbb{C}^{\mathrm{st}}$ is a trace such that* $\mathrm{initial}(\mathbb{C}) \xrightarrow{s} n$ *for an* $n = (\{(\bar{t}, h) : E\}, \overline{m})$*, then there exists a permutation $\pi_{\mathbb{P}}$ over $\mathbb{P}$ such that the following holds:*

1. *The heap h is a copycat heap for s over $\mathfrak{A} \Rightarrow \mathfrak{A}'$.*
2. *The set of messages that n can immediately send,* $\mathrm{ready}(n)$*, is exactly the set of messages p such that $s = s_1{::}o{::}s_2$ and $p \notin s_2$ where the form of o and p is $o = (\mathbf{O}, (a, p, p', d))$ and $p = (\mathbf{P}, (\tilde{\pi}_{\mathbb{A}}(a), \tilde{\pi}_{\mathbb{P}}(p), \tilde{\pi}_{\mathbb{P}}(p'), d))$ (i.e. the "copy" of o).*

Proof. Induction on the length of s. The base case is immediate.

We need to show that if the theorem holds for a trace s, then it also holds for $s{::}\alpha$. We thus assume that there exists a permutation $\pi_{\mathbb{P}}$ such that the hypothesis holds for s and that *initial*$(\mathbb{C}) \xrightarrow{s} n \to^* \xrightarrow{\alpha} n'$.

1. If $\alpha = (\mathbf{P}, (\tilde{\pi}_{\mathbb{A}}(a), \tilde{\pi}_{\mathbb{P}}(p), \tilde{\pi}_{\mathbb{P}}(p'), d))$ then by (2) there must be a message $o = (\mathbf{O}, (a, p, p', d))$ such that $s = s_1{::}o{::}s_2$ and $\alpha \in$ *ready*(n). Since we "chose" $\pi_{\mathbb{P}}$ such that p can only be gotten from the thread spawned by o, we can proceed by cases as we did Theorem 3.17 to see that the heap structure is correct in each case.

2.
   - If $\alpha = (\mathbf{P}, (\tilde{\pi}_{\mathbb{A}}(a), \tilde{\pi}_{\mathbb{P}}(p), \tilde{\pi}_{\mathbb{P}}(p'), d))$ then by (2) there must be a message $o = (\mathbf{O}, (a, p, p', d))$ such that $s = s_1{::}o{::}s_2$ and $\alpha \in$ *ready*(n). By Lemma 3.21, *ready*$(n) =$ *ready*$(n') \cup \{\alpha\}$. We can easily verify that (2) holds for n'.
   - If $\alpha = (\mathbf{O}, (a, p_1, p_1', d))$, then we can proceed as in Theorem 3.17 to see that a message $p = (\mathbf{P}, (\tilde{\pi}_{\mathbb{A}}(a), p_2, p_2', d)) \in$ *ready*(n'). We then simply construct our extended permutation such that the hypothesis holds.

□

3.4 Composition

The definition of composition in Hyland-Ong games [19] is eerily similar to our definition of trace composition, so we might expect HRAM net composition to correspond to it. That is, however, only superficially true: the nominal setting that we are using [9] brings to light what happens to the justification pointers in composition.

If A is an interface, $s \in \mathit{traces}_A$ and $X \subseteq \mathit{sup}(A)$, we define the *reindexing deletion* operator $s \downharpoonleft X$ as follows, where $(s', \rho) = s \downharpoonleft X$ inductively:

$$\begin{aligned} \epsilon \downharpoonleft X &\triangleq (\epsilon, id) \\ s{::}(l, (a, p, p', d)) \downharpoonleft X &\triangleq (s'{::}(l, (a, \rho(p), p', d)), \rho) && \text{if } a \notin X \\ s{::}(l, (a, p, p', d)) \downharpoonleft X &\triangleq (s', \rho \cup \{p' \mapsto \rho(p)\}) && \text{if } a \in X \end{aligned}$$

We write $s \downharpoonleft X$ for s' when $s \downharpoonleft X = (s', \rho)$ in the following definition:

Definition 3.23. *The* game composition *of the sets of traces* $S_1 \subseteq \mathrm{traces}_{A \Rightarrow B}$ *and* $S_2 \subseteq \mathrm{traces}_{B' \Rightarrow C}$ *with* $\pi \vdash B =_{\mathbb{A}} B'$ *is*

$$S_1 ;_{\mathfrak{G}} S_2 \triangleq \{ s \downharpoonleft B \mid s \in \mathrm{traces}_{A \otimes B \otimes C} \wedge s \downharpoonleft C \in S_1 \wedge \pi \cdot s^{*B} \downharpoonleft A \in S_2 \}$$

Clearly we have $S_1 ; S_2 \neq S_1 ;_{\mathfrak{G}} S_2$ for sets of traces S_1 and S_2, which reinforces the practical problem in the beginning of this section.

Composition is constructed out of three copycat-like behaviours, as sketched in Fig. 6 for a typical play at some types A,B and C. As a trace in the nominal model, this is:

$$\begin{aligned} &(q6, p0, p1){::}(q4, p1, p2){::}(q3, p2, p3){::} \\ &\qquad (q2, p1, p4){::}(q1, p4, p5){::}(q5, p1, p6){::}(a5, p6){::} \\ &\qquad\qquad (a1, p5){::}(a2, p4){::}(a3, p3){::}(a4, p2){::}(a6, p1) \end{aligned}$$

We see that this *almost* corresponds to three interleaved copycats as described above; between A, B, C and A', B', C'. There is, however, a small difference: The move q_1, if it were to blindly follow the recipe of a copycat, would dereference the pointer p_4, yielding p_3, and so incorrectly make the move q_5 justified by q_3, whereas it really should be justified by q_6 as in the diagram. This is precisely the problem explained at the beginning of this section.

To make a pointer *extension*, when the B-initial move q_3 is performed, it should map p_4 not only to p_3, but also to the pointer that p_2 points to, which is p_1 (the dotted line in the diagram). When the A-initial move q_1 is performed, it has access to both of these pointers that p_4 maps to, and can correctly make the q_5 move by associating it with pointers p_1 and a fresh p_6.

Let $\mathfrak{A}'$, $\mathfrak{B}'$, and $\mathfrak{C}'$ be game interfaces such that $\pi_{\mathfrak{A}} \vdash \mathfrak{A} =_{\mathbb{A}} \mathfrak{A}'$, $\pi_{\mathfrak{B}} \vdash \mathfrak{B} =_{\mathbb{A}} \mathfrak{B}'$,

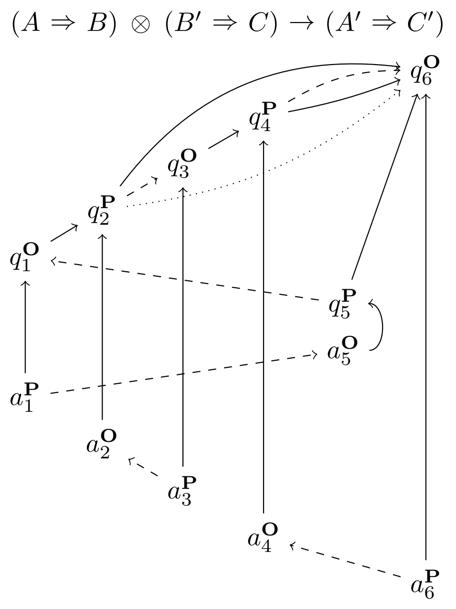

Figure 6: Composition from copycat

$\pi_{\mathfrak{C}} \vdash \mathfrak{C} =_{\mathbb{A}} \mathfrak{C}'$, and

$$\begin{aligned}
(A' \Rightarrow A, P_A) &= \mathcal{C}_{\texttt{exq},\pi_{\mathfrak{A}}^{-1},\mathfrak{A}'}\\
(B \Rightarrow B', P_B) &= \mathcal{C}_{\texttt{exi},\pi_{\mathfrak{B}},\mathfrak{B}}\\
(C \Rightarrow C', P_C) &= \mathcal{C}_{\texttt{cci},\pi_{\mathfrak{C}},\mathfrak{C}},\ \text{where}
\end{aligned}$$

$$\texttt{exi} \triangleq 0,3 \leftarrow \texttt{get}\ 0;\ 1 \leftarrow \texttt{new}\ 1,0$$

$$\texttt{exq} \triangleq \emptyset,0 \leftarrow \texttt{get}\ 0;\ 1 \leftarrow \texttt{new}\ 1,3$$

Then the game composition operator $K_{\mathfrak{A},\mathfrak{B},\mathfrak{C}}$ is:

$$K_{\mathfrak{A},\mathfrak{B},\mathfrak{C}} \triangleq ((A \Rightarrow B) \otimes (B' \Rightarrow C) \Rightarrow (A' \Rightarrow C'), P_A \cup P_B \cup P_C).$$

Using the game composition operator K we can define GAM-net composition using **HRAMnet** compact closed combinators. Let $f : \mathfrak{A} \Rightarrow \mathfrak{B}, g : \mathfrak{B} \Rightarrow \mathfrak{C}$ be GAM-nets. Then their composition is defined as

$$\begin{aligned}
&f;_{GAM}\, g \triangleq \Lambda_A^{-1}(\Lambda_A(f) \otimes \Lambda_B(g)); K_{\mathfrak{A},\mathfrak{B},\mathfrak{C}})), \text{where}\\
&\Lambda_A(f : A \to B) \triangleq (\eta_A; (id_{A^*} \otimes f)) : I \to A^* \otimes B\\
&\Lambda_A^{-1}(f : I \to A \otimes B) \triangleq ((id_A \otimes f); (\varepsilon_A \otimes id_B)) : A \to B.
\end{aligned}$$

Composition is represented diagrammatically as in Fig. 7. Note the comparison with the naive composition from Fig. 4. HRAMs f and g are not plugged in directly, although the interfaces match. Composition is mediated by the operator K, which preserves the locality of freshly generated names, exchanging non-local pointer names with local pointer names and storing the mapping between the two as copy-cat links, indicated diagrammatically by dotted lines in K.

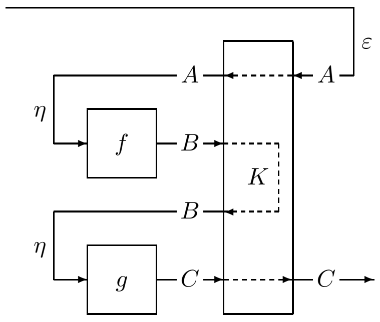

Figure 7: Composing GAMs using the K HRAM

Theorem 3.24. *If $f : \mathfrak{A} \to \mathfrak{B}$ and $g : \mathfrak{B}' \to \mathfrak{C}$ are game nets such that $\pi_{\mathfrak{B}} \vdash \mathfrak{B} =_{\mathbb{A}} \mathfrak{B}'$, f implements $S_f \subseteq P_{\mathfrak{A}\Rightarrow\mathfrak{B}}$, and g implements $S_g \subseteq P_{\mathfrak{B}'\Rightarrow\mathfrak{C}}$, then $f;_{GAM}\, g$ implements $(S_f;_{\mathfrak{G}}\, S_g)$.*

Definition 3.25. *If s is a trace, h is a heap, $\mathfrak{A}$ is a game interface, and $\pi_{\mathbb{P}}$ is a permutation over $\mathbb{P}$, we say that h is an* extended copycat heap *for s over $\mathfrak{A}$ if and only if:*

1. *For every pending $\mathbf{P}$-question non-initial in $\mathfrak{A}$ in s, i.e. $(\mathbf{P}, (a, p, p', d)) \subseteq s$ ($a \in \mathrm{qst}_{\mathfrak{A}} \setminus \mathrm{ini}_{\mathfrak{A}}$), $h(p') = (\tilde{\pi}_{\mathbb{P}}(p'), \emptyset)$.*
2. *For every pending $\mathbf{P}$-question initial in $\mathfrak{A}$ in s and its justifying move, i.e. $(\mathbf{O}, (a_1, p_1, p, d_1))::s'::(\mathbf{P}, (a_2, p, p_2, d_2)) \subseteq s$ ($a_2 \in \mathrm{ini}_{\mathfrak{A}}$), $h(p_2) = (\tilde{\pi}_{\mathbb{P}}(p_2), \tilde{\pi}_{\mathbb{P}}(p_1))$.*

Theorem 3.26. *If $f : \mathfrak{A} \to \mathfrak{B}$ and $g : \mathfrak{B}' \to \mathfrak{C}$ are game nets such that $\pi_{\mathfrak{B}} \vdash \mathfrak{B} =_{\mathbb{A}} \mathfrak{B}'$, f implements $S_f \subseteq P_{\mathfrak{A}\Rightarrow\mathfrak{B}}$, and g implements $S_g \subseteq P_{\mathfrak{B}'\Rightarrow\mathfrak{C}}$, then $(S_f;_{\mathfrak{G}}\, S_g)^{\mathrm{st,alt}} \subseteq_{\mathbb{AP}} [\![f;_{GAM}\, g]\!] = [\![\Lambda_A^{-1}(\Lambda_A(f) \otimes \Lambda_{B'}(g); K_{\mathfrak{A},\mathfrak{B},\mathfrak{C}})]\!]$.*

Proof. We show that $s' \in (S_f;_{\mathfrak{G}}\, S_g)^{st,alt}$ implies that there exists a $\pi_{\mathbb{P}}$ such that $\pi_{\mathfrak{A},\mathfrak{C}} \cdot \pi_{\mathbb{P}} \cdot s' \in [\![f;_{GAM}\, g]\!] = [\![\Lambda_A^{-1}(\Lambda_A(f) \otimes \Lambda_{B'}(g); K_{\mathfrak{A},\mathfrak{B},\mathfrak{C}})]\!] = [\![\Lambda_A(f)]\!] \otimes [\![\Lambda_{B'}(g)]\!]; [\![K_{\mathfrak{A},\mathfrak{B},\mathfrak{C}}]\!]$. Recall the definition of game composition:

$$S_f;_{\mathfrak{G}}\, S_g \triangleq \{s \upharpoonright B \mid s \in traces_{A\otimes B\otimes C} \wedge s \upharpoonright C \in S_f \wedge \pi_{\mathfrak{B}} \cdot s^{*B} \upharpoonright A \in S_g\}$$

We proceed by induction on the length of such an s:

- Hypothesis. There exists an s_K such that $initial(K_{\mathfrak{A},\mathfrak{B},\mathfrak{C}}) \xrightarrow{s_K} n$ where $n = (\{(\emptyset, h) : E\}, \emptyset)$ and h is exactly (nothing more than) the union of a copycat heap for s_K over $\mathfrak{A}' \Rightarrow \mathfrak{A}$, a copycat heap for s_K over $\mathfrak{C} \Rightarrow \mathfrak{C}'$ and an extended copycat heap for s_K over $\mathfrak{B} \Rightarrow \mathfrak{B}'$.

Let

$$\begin{aligned}
s_f &\triangleq s \upharpoonright C \\
s_g &\triangleq \pi_{\mathfrak{B}} \cdot s^{*B} \upharpoonright A \\
s_{f;g} &\triangleq s \upharpoonright B \\
s_{Kf} &\triangleq s_K - A', B', C, C', \text{ the part of } s_K \text{ relating to } f \\
s_{Kg} &\triangleq s_K - A, A', B, C', \text{ the part of } s_K \text{ relating to } g \\
s_{Kf;g} &\triangleq s_K - A, B, B', C, \text{ the part of } s_K \text{ relating to the whole game net.}
\end{aligned}$$

We require that s_K fulfils $s^*_{Kf} = s_f$, $s^*_{Kg} = s_g$, and $s_{Kf;g} = \pi_{\mathfrak{A},\mathfrak{C}} \cdot \pi_{\mathbb{P}} \cdot s_{f;g}$. Note that $s_{Kf;g}$ is the trace of $f;_{GAM}\, g$, by the definition of trace composition.

- Base case. Immediate.
- Inductive step. Assume $s = s'{::}\alpha$ and that the hypothesis holds for s' and some $\pi'_{\mathbb{P}}$ and s'_K. We proceed by cases on the α message:
  - If $\alpha = (\mathbf{O}, (a, p, p', d))$, we have three cases:
    - If $a \in \mathit{sup}(A)$, intuitively this means that we are getting a message from outside the K engine, and need to propagate it through K to f. We construct s_K and $\pi_{\mathbb{P}}$, such that $s_K = s'_K{::}(\mathbf{O}, (\pi_{\mathfrak{A}}(a), \tilde{\pi}_{\mathbb{P}}(p), \tilde{\pi}_{\mathbb{P}}(p'), d)){::}\alpha^*$, by further sub-cases on a ($\pi_{\mathbb{P}}$ will be determined by steps of the K configuration):
      - · $a \in \mathit{ini}_{\mathfrak{A}}$ cannot be the case because an initial message in A must be justified by an initial ($\mathbf{O}$-message) in C, and so must be a $\mathbf{P}$-message.
      - · If $a \in (\mathit{qst}_{\mathfrak{A}} \setminus \mathit{ini}_{\mathfrak{A}}) \cup \mathit{ans}_{\mathfrak{A}}$, this means that $s' \upharpoonright C{::}\alpha = (s'{::}\alpha) \upharpoonright C$ as the message must be justified by a message from $\mathfrak{A}$. As f is $\mathbf{O}$-closed $s \upharpoonright C \in [\![\Lambda_A(f)]\!]$. This trace can be stepped to by n' just like how it was done in Theorem 3.17. We can verify that the parts of the hypothesis not in that theorem hold – in particular for this case we have $s_{Kf} = s'_{Kf}{::}\alpha^*$, so indeed $s^*_{Kf} = s_f$ as required.
    - $a \in \mathit{sup}(B)$:
      Intuitively this means that g is sending a message to f, which has to go through K. We construct s_K and $\pi_{\mathbb{P}}$, such that $s_K = s'_K{::}(\mathbf{O}, (a, \tilde{\pi}_{\mathbb{P}}(p), \tilde{\pi}_{\mathbb{P}}(p'), d)){::}\pi_{\mathfrak{B}} \cdot \alpha^*$, by further sub-cases on a ($\pi_{\mathbb{P}}$ will be determined by steps of the K configuration):
      - · If $a \in \mathit{ini}_{\mathfrak{B}}$, there must be a pending $\mathbf{P}$-message from $\mathfrak{C}$ justifying α in s', i.e. $(\mathbf{P}, (a_0, p_0, \tilde{\pi}_{\mathbb{P}}(p), d_0)) \subseteq s'$ and then by Definition 3.20 $h(\tilde{\pi}_{\mathbb{P}}(p)) = (p, \emptyset)$ (as $\tilde{\pi}_{\mathbb{P}}$ is its own inverse). This means that (running the `exi` instruction) we get:

$$n' \xrightarrow{(\mathbf{O},(\pi_{\mathfrak{B}}(a),\tilde{\pi}_{\mathbb{P}}(p),\tilde{\pi}_{\mathbb{P}}(p'),d))} \longrightarrow^* \xrightarrow{\alpha^*}$$
$$(\{(\emptyset, h \cup \{p' \mapsto (\tilde{\pi}_{\mathbb{P}}(p'), p)\}) \;:\; E\}, \emptyset) = n$$

        Now $\pi_{\mathfrak{B}} \cdot \alpha^*$ is a new pending **P**-question in the trace that is initial in $\mathfrak{B} \Rightarrow \mathfrak{B}'$, but our new heap mapping fulfils clause (2) of Definition 3.25 as required.

      - If $a \in (qst_{\mathfrak{B}} \setminus ini_{\mathfrak{B}}) \cup ans_{\mathfrak{B}}$, this is similar to the $\mathfrak{A}$ case (note that the extended copycat only differs from the ordinary copycat for initial messages).

    - If $a \in sup(C)$.
      Intuitively this means that we are getting a message from outside the K engine, and need to propagate it through K to g. We construct s_K and $\pi_{\mathbb{P}}$, such that:

$$s_K = s'_K{::}(\mathbf{O}, (\pi_{\mathfrak{C}}(a), \tilde{\pi}_{\mathbb{P}}(p), \tilde{\pi}_{\mathbb{P}}(p'), d)){::}\alpha^*$$

      In this case, the code that we will run is just that of $\mathbb{C}$, so we can proceed like in Theorem 3.17, easily verifying our additional assumptions.

  - If $\alpha = (\mathbf{P}, (a, p, p', d))$, we have three cases:

    - If $a \in sup(A)$, intuitively this means that we get a message from f and need to propagate it through K to the outside. By further sub-cases on a, we construct s_K and $\pi_{\mathbb{P}}$, such that:

$$s_K = s'_K{::}\alpha^*{::}(\mathbf{P}, (\pi_{\mathfrak{A}}(a), \tilde{\pi}_{\mathbb{P}}(p), \tilde{\pi}_{\mathbb{P}}(p'), d))$$

      The pointer permutation $\pi_{\mathbb{P}}$ will be determined by steps of the K configuration.

      - If $a \in ini_{\mathfrak{A}}$, then α must be justified in s' by a pending and initial **P**-question from $\mathfrak{B}$ by the definition of $\mathfrak{A} \Rightarrow \mathfrak{B}$ which must in turn be justified by a pending and initial **O**-question from $\mathfrak{C}$ by the definition of $\mathfrak{B} \Rightarrow \mathfrak{C}$. In s'_K, we have (since $s'_{K f;g} = \pi_{\mathfrak{A},\mathfrak{C}} \cdot \pi_{\mathbb{P}} \cdot s'_{f;g}$)

$$s'_K = s_1{::}(\mathbf{O}, (a_{\mathfrak{C}'}, p_0, p_{\mathfrak{C}'}, d_{\mathfrak{C}'})){::}s_2{::}(\mathbf{P}, (a_{\mathfrak{B}}, p_{\mathfrak{C}'}, p, d_{\mathfrak{C}'})){::}s_3$$

        This means that clause (2) in Definition 3.25 applies, such that $h(p) = (\tilde{\pi}_{\mathbb{P}}(p), \tilde{\pi}_{\mathbb{P}}(p_0))$ and that (running the `exq` instruction) we get:

$$n' \xrightarrow{\alpha^*} \to^* \xrightarrow{(\mathbf{P},(\pi_{\mathfrak{A}}(a),\tilde{\pi}_{\mathbb{P}}(p),\tilde{\pi}_{\mathbb{P}}(p'),d))}$$
$$(\{(\emptyset, h \cup \{\tilde{\pi}_{\mathbb{P}}(p') \mapsto (p', d)\}) \ : \ E\}, \emptyset) = n$$

        Clause (1) of Definition 3.25 applies to these new messages and trivially holds.

      - When $a \in (qst_{\mathfrak{A}} \setminus ini_{\mathfrak{A}}) \cup ans_{\mathfrak{A}}$, the code that we will run is just that of $\mathbb{C}$, so we can proceed like in Theorem 3.17, also verifying our additional assumptions.

    - If $a \in sup(B)$, intuitively this means that f is sending a message to g, which has to go through K.

      - $a \in ini_{\mathfrak{B}}$ cannot be the case for a **P**-message.

  · When $a \in (qst_{\mathfrak{B}} \setminus ini_{\mathfrak{B}}) \cup ans_{\mathfrak{B}}$, the code that we will run is just that of $\mathbb{C}$, so we can proceed like in Theorem 3.17, also verifying our additional assumptions.

∗ If $a \in sup(C)$, intuitively this means that we get a message from g and need to propagate it through K to the outside.

  · $a \in ini_{\mathfrak{C}}$ cannot be the case for a **P**-message.

  · When $a \in (qst_{\mathfrak{C}} \setminus ini_{\mathfrak{C}}) \cup ans_{\mathfrak{C}}$, the code that we will run is just that of $\mathbb{C}$, so we can proceed like in Theorem 3.17, also verifying our additional assumptions.

□

Lemma 3.27. *If $f : \mathfrak{A} \to \mathfrak{B}$ and $g : \mathfrak{B}' \to \mathfrak{C}$ are game nets such that $\pi_{\mathfrak{B}} \vdash \mathfrak{B} =_{\mathbb{A}} \mathfrak{B}'$, f implements $S_f \subseteq P_{\mathfrak{A}\Rightarrow\mathfrak{B}}$, and g implements $S_g \subseteq P_{\mathfrak{B}'\Rightarrow\mathfrak{C}}$, then $[\![(f ;_{GAM} g)]\!]$ is* **P***-closed with respect to $(S_f ;_{\mathfrak{G}} S_g)$.*

Proof. Similar to Theorems 3.22 and 3.26. We identify the set $ready(n)$ with "uncopied" messages of a K net configuration n and show that these are legal according to the game composition. Then we show by induction that, assuming a heap as in Theorem 3.26, the $ready(n)$ set is precisely those messages. □

3.5 Diagonal

For game interfaces $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{A}_3$ and permutations π_{ij} such that $\pi_{ij} \vdash \mathfrak{A}_i =_{\mathbb{A}} \mathfrak{A}_j$ for $i \neq j \in \{1, 2, 3\}$, we define the family of diagonal engines as:

$$\delta_{\pi_{12},\pi_{13},\mathfrak{A}} = (A_1 \Rightarrow A_2 \otimes A_3, P_1 \otimes P_2 \otimes P_3)$$

where, for $i \in \{2, 3\}$,

$$\begin{aligned}
P_1 \triangleq\ & \{q_1 \mapsto \texttt{ccq};\ \texttt{ifzero}\ 3\ (\texttt{spark}\, q_2)\ (\texttt{spark}\, q_3) \\
& \qquad \mid q_1 \in opp_{\mathfrak{A}_1} \cap qst_{\mathfrak{A}_1} \wedge q_2 = \pi_{12}(q_1) \wedge q_3 = \pi_{13}(q_1)\} \\
& \cup \{a_1 \mapsto \texttt{cca};\ \texttt{ifzero}\ 3\ (\texttt{spark}\, a_2)\ (\texttt{spark}\, a_3) \\
& \qquad \mid a_1 \in opp_{\mathfrak{A}_1} \cap ans_{\mathfrak{A}_1} \wedge a_2 = \pi_{12}(a_1) \wedge a_3 = \pi_{13}(a_1)\} \\
P_i \triangleq\ & \{q_i \mapsto 3 \leftarrow \texttt{set}\ (i-2);\ \texttt{cci};\ \texttt{spark}\, q_1 \mid q_i \in ini_{\mathfrak{A}_i} \wedge q_1 = \pi_{1i}^{-1}(q_i)\} \\
& \cup \{q_i \mapsto \texttt{ccq};\ \texttt{spark}\, q_1 \mid q_i \in (opp_{\mathfrak{A}_i} \cap qst_{\mathfrak{A}_i}) \setminus ini_{\mathfrak{A}_i} \wedge q_1 = \pi_{1i}^{-1}(q_i)\} \\
& \cup \{a_i \mapsto \texttt{cca};\ \texttt{spark}\, a_1 \mid a_i \in opp_{\mathfrak{A}_i} \cap ans_{\mathfrak{A}_i} \wedge a_1 = \pi_{1i}^{-1}(a_i)\}.
\end{aligned}$$

The diagonal is almost identical to the copycat, except that an integer value of 0 or 1 is associated, in the heap, with the name of each message arriving on the A_2 and A_3 interfaces (hence the `set` statements, to be used for routing back messages arriving on A_1 using `ifzero` statements). By abuse of notation, we also write δ for the net $singleton(\delta)$.

Lemma 3.28. *The δ net is the diagonal net, i.e. $[\![\delta_{\pi_{12},\pi_{23},\mathfrak{A}}; \Pi_i]\!] = [\![\mathbb{C}_{\pi_i,\mathfrak{A}}]\!]$.*

Proof. We show that $s \in [\![\delta_{\pi_{12},\pi_{23}}; \Pi_1]\!]$ implies $s \in [\![\mathbb{C}_{\pi_{12},\mathfrak{A}_1,\mathfrak{A}_2}]\!]$ and the converse (the Π_2 case is analogous), by induction on the trace length. There is a simple relationship between the heap structures of the respective net configurations — they have the same structure but the diagonal stores additional integers for identifying what "side" a move comes from. □

3.6 Fixpoint

We define a family of GAMs $Fix_{\mathfrak{A}}$ with interfaces $(\mathfrak{A}_1 \Rightarrow \mathfrak{A}_2) \Rightarrow \mathfrak{A}_3$ where there exist permutations $\pi_{i,j}$ such that $\pi_{i,j} \vdash \mathfrak{A}_i =_{\mathbb{A}} \mathfrak{A}_j$ for $i \neq j \in \{1,2,3\}$. The fixpoint engine is defined as $Fix_{\pi_{12},\pi_{13},\mathfrak{A}} = \Lambda_A^{-1}(\delta_{\pi_{12},\pi_{13},\mathfrak{A}})$.

Let $\mathit{fix}_{\pi_{12},\pi_{13},\mathfrak{A}} : (\mathfrak{A} \Rightarrow \pi_{12} \cdot \mathfrak{A}) \Rightarrow \pi_{13} \cdot \mathfrak{A}$ be the game-semantic strategy for fixpoint in Hyland-Ong games [19, p. 364].

Theorem 3.29. *$Fix_{\pi_{12},\pi_{13},\mathfrak{A}}$ implements $\mathit{fix}_{\pi_{12},\pi_{13},\mathfrak{A}}$.*

The proof of this is immediate considering the three cases of moves from the definition of the game-semantic strategy. It is interesting to note here that we "*force*" a HRAM with interface $A_1 \Rightarrow A_2 \otimes A_3$ into a GAM with game interface $(\mathfrak{A}_3 \Rightarrow \mathfrak{A}_1) \Rightarrow \mathfrak{A}_2$, which has underlying interface $(A_3 \Rightarrow A_1) \Rightarrow A_2$. In the **HRAMnet** category, which is symmetric compact-closed, the two interfaces are isomorphic (with $A_1^* \otimes A_2 \otimes A_3$), but as game interfaces they are not. It is rather surprising that we can reuse our diagonal GAMs in such brutal fashion: in the game interface for fixpoint there is a reversed enabling relation between A_3 and A_1. The reason why this still leads to legal plays only is because the onus of producing the justification pointers in the initial move for A_3 lies with the Opponent, which cannot exploit the fact that the diagonal is "wired illegally". It only sees the fixpoint interface and must play accordingly. It is fair to say that that fixpoint interface is more restrictive to the Opponent than the diagonal interface, because the diagonal interface allows extra behaviours, e.g. sending initial messages in A_3, which are no longer legal.

3.7 Other ICA constants

A GAM net for an integer literal n can be defined using the following engine (whose interface corresponds to the ICA `exp` type).

$$\begin{aligned} \mathit{lit}_n &\triangleq (\{(\mathbf{O}, q), (\mathbf{P}, a)\}, P), \text{ where} \\ P &\triangleq \{q \mapsto \texttt{flip}\ 0,1;\ 1 \leftarrow \texttt{set}\ \emptyset;\ 2 \leftarrow \texttt{set}\ n;\ \texttt{spark}\ a\} \end{aligned}$$

We see that upon getting an input question on port q, this engine will respond with a legal answer containing n as its value (register 2).

The conditional at type `exp` can be defined using the following engine, with the convention that $\{(\mathbf{O}, q_i), (\mathbf{P}, a_i)\} = \mathtt{exp}_i$.

$$
\begin{aligned}
&\mathit{if} \triangleq (\mathtt{exp}_1 \Rightarrow \mathtt{exp}_2 \Rightarrow \mathtt{exp}_3 \Rightarrow \mathtt{exp}_4, P), \text{ where}\\
&P \triangleq \{q_4 \mapsto \mathtt{cci};\ \mathtt{spark}\ q_1,\\
&\quad a_1 \mapsto \mathtt{cca};\ \mathtt{flip}\ 0,1;\ \mathtt{cci};\ \mathtt{ifzero}\ 2\ (\mathtt{spark}\ q_3)\ (\mathtt{spark}\ q_2),\\
&\quad a_2 \mapsto \mathtt{cca};\ \mathtt{spark}\ a_4,\\
&\quad a_3 \mapsto \mathtt{cca};\ \mathtt{spark}\ a_4\}
\end{aligned}
$$

We can also define primitive operations, e.g. $+ : \mathtt{exp} \Rightarrow \mathtt{exp} \Rightarrow \mathtt{exp}$, in a similar manner. An interesting engine is that for *newvar*:

$$
\begin{aligned}
&\mathit{newvar} \triangleq ((\mathtt{exp}_1 \otimes (\mathtt{exp}_2 \Rightarrow \mathtt{com}_3) \Rightarrow \mathtt{exp}_4) \Rightarrow \mathtt{exp}_5, P)\\
&\quad P \triangleq \{q_5 \mapsto 3 \leftarrow \mathtt{set}\ 0;\ \mathtt{cci};\ \mathtt{spark}\ q_4,\\
&\qquad q_1 \mapsto \emptyset, 2 \leftarrow \mathtt{get}\ 0;\ \mathtt{flip}\ 0,1;\ 1 \leftarrow \mathtt{set}\ \emptyset;\ \mathtt{spark}\ a_1,\\
&\qquad q_3 \mapsto \mathtt{flip}\ 0,1;\ 1 \leftarrow \mathtt{new}\ 0,1;\ \mathtt{spark}\ q_2,\\
&\qquad a_2 \mapsto \emptyset, 3 \leftarrow \mathtt{get}\ 0;\ \mathtt{update}\ 3\ 2;\ \mathtt{cca};\ \mathtt{spark}\ a_3,\\
&\qquad a_4 \mapsto \mathtt{cca};\ \mathtt{spark}\ a_5\}
\end{aligned}
$$

We see that we store the variable in the second component of the justification pointer that justifies q_4, so that it can be accessed in subsequent requests. A slight problem is that moves in $\mathtt{exp}_2$ will actually not be justified by this pointer which we remedy in the q_3 case, by storing a pointer to the pointer with the variable as the second component of the justifier of q_2, which means that we can access and update the variable in a_2.

We can easily extend the HRAMs with new instructions to interpret parallel execution and semaphores, but we omit them from the current presentation.

4 Seamless distributed compilation for ICA

4.1 The language ICA

ICA is PCF extended with constants to facilitate local effects. Its ground types are expressions and commands ($\mathsf{exp}, \mathsf{com}$), with the type of assignable variables desugared as $\mathsf{var} \triangleq \mathsf{exp} \times (\mathsf{exp} \to \mathsf{com})$. Dereferencing and assignment are desugared as the first, respectively second, projections from the type of assignable variables. The local variable binder is $\mathsf{new} : (\mathsf{var} \to \mathsf{com}) \to \mathsf{com}$. ICA also has a type of split binary semaphores $\mathsf{sem} \triangleq \mathsf{com} \times \mathsf{com}$, with the first and second projections corresponding to $\mathsf{set}, \mathsf{get}$, respectively (see [14] for the full definition, including the game-semantic model).

In this section we give a compilation method for ICA into GAM nets. The compilation is compositional on the syntax and it uses the constructs of the previous section. ICA types are compiled into GAM interfaces which correspond to their game-semantic arenas in the obvious way. We will use $A, B, \ldots$ to refer to an ICA type and to the GAM interface. Sec. 3 has already developed all the infrastructure needed to interpret the constants of ICA (Sec. 3.7), including

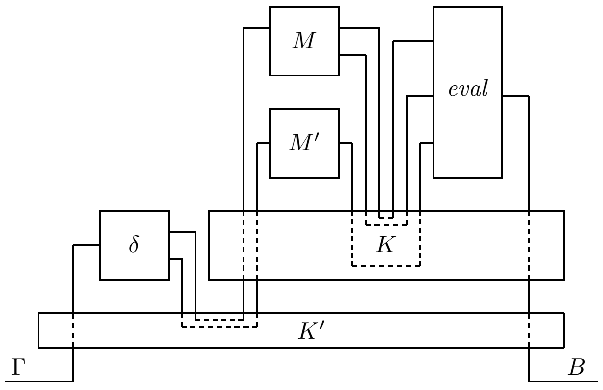

Figure 8: GAM net for application

fixpoint (Sec. 3.6). Given an ICA type judgment $\Gamma \vdash M : A$ with Γ a list of variable-type assignments $x_i : A_i$, M a term and A a type, a GAM implementing it G_M is defined compositionally on the syntax as follows:

$$\begin{aligned} G_{\Gamma\vdash MM':A} &= \delta_{\pi_1,\pi_2,\Gamma};_{GAM}\ (G_{\Gamma\vdash M:A\to B} \otimes G_{\Gamma\vdash M':B});_{GAM}\ eval_{A,B} \\ G_{\Gamma\vdash\lambda x:A.M:A\to B} &= \Lambda_A(G_{\Gamma,x:A\vdash M:B}) \\ G_{x:A,\Gamma\vdash x:A} &= \Pi_{\mathfrak{G}A}; \mathbb{C}_{A,\pi}, \end{aligned}$$

Where $eval_{A,B} \triangleq \Lambda_B^{-1}(\mathbb{C}_{A\Rightarrow B,\pi})$ for a suitably chosen port renaming π and $\Pi_{\mathfrak{G}A}$ and $\Pi_{\mathfrak{G}1}$ and $\Pi_{\mathfrak{G}2}$ are HRAMs with signatures $\Pi_{\mathfrak{G}i} = (A_1 \otimes A_2 \Rightarrow A_3, P_i)$ such that they copycat between A_3 and A_i and ignore $A_{j\neq i}$. The interpretation of function application, which is the most complex, is shown diagrammatically in Fig. 8. The copycat connections are shown using dashed lines.

Theorem 4.1. *If M is an ICA term, G_M is the GAM implementing it and σ_M its game-semantic strategy then G_M implements σ_M.*

The correctness of compilation follows directly from the correctness of the individual GAM nets and the correctness of GAM composition $;_{GAM}$.

4.2 Prototype implementation

Following the recipe in the previous section we can produce an implementation of any ICA term as a GAM net. GAMs are just special-purpose HRAMs, with no special operations. HRAMs, in turn, can easily be implemented on any conventional computer with the usual store, control and communication facilities. A GAM net is also just a special-purpose HRAM net, which is a powerful abstraction of communication processes, as it subsumes through the `spark` instruction communication between processes (threads) on the same physical machine or located on distinct physical machines and communicating via a point-to-point network. We have built a prototype compiler based on GAMs by implementing them in C, managing processes using standard UNIX threads and physical network distribution using MPI [17].[2]

[2]Download with source code from `http://veritygos.org/gams`.

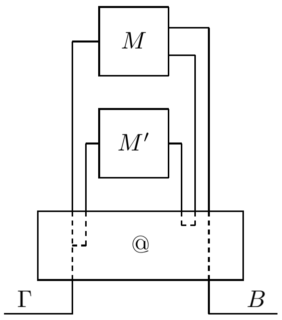

Figure 9: Optimised GAM net for application

The actual distribution is achieved using light pragma-like code annotations. In order to execute a program at node A but delegate one computation to node B and another computation to node C we simply annotate an ICA program with node names, e.g.:

```
{new x. x := {f(x)}@B + {g(x)}@C; !x}@A
```

Note that this gives node B, via function f, read-write access to memory location x which is located at node A. Accessing non-local resources is possible, albeit possibly expensive.

Several facts make the compilation process quite remarkable:

- It is *seamless* (in the sense of [8]), allowing distributed compilation where communication is never explicit but always realised through function calls.
- It is *flexible*, allowing any syntactic sub-term to be located at any designated physical location, with no impact on the semantics of the program. The access of *non-local* resources is always possible, albeit possibly at a cost (latency, bandwidth, etc.).
- It is *dynamic*, allowing the relocation of GAMs to different physical nodes at run time. This can be done with extremely low overhead if the GAM heap is empty.
- It does not require any form of *garbage collection*, even on local nodes, although the language combines (ground) state, higher-order functions and concurrency. This is because a pointer associated with a pointer is not needed if and only if the question is answered; then it can be safely deallocated.

The current implementation does not perform any optimisations, and the resulting code is inefficient. Looking at the implementation of application in Fig. 8 it is quite clear that a message entering the GAM net via port A needs to undergo four pointer renamings before reaching the GAM for M. This is the cost we pay for compositionality. However, the particular configuration for application can be significantly simplified using standard peephole optimisation, and we can reach the much simpler, still correct implementation in Fig. 9. Here the functionality of the two compositions, the diagonal, and the *eval* GAMs have been combined and optimised into a single GAM, requiring only one pointer renaming before reaching M. Other optimisations can be introduced to simplify GAM nets, in particular to obviate the need for the use of composition GAMs K, for example by showing that composition of first-order closed terms (such as those used for most constants) can be done directly.

5 Conclusions, further work

In a previous paper we have argued that distributed and heterogeneous programming would benefit from the existence of architecture-agnostic, seamless compilation methods for conventional programming languages which can allow the programmer to focus on solving algorithmic problems without being overwhelmed by the minutiae of driving complex computational systems [8]. In *loc. cit.* we give such a compiler for PCF, based directly on the Geometry of Interaction. In this paper we show how Game Semantics can be expressed operationally using abstract machines very similar to networked conventional computers, a further development of the IAM/JAM game machines. We believe any programming language with a semantic model expressed as Hyland-Ong-style pointer games [19] can be readily represented using GAMs and then compiled to a variety of platforms such as MPI. Even more promising is the possible leveraging of more powerful infrastructure for distributed computing that can mask much of the complexities of distributed programming, such as fault-tolerance [23].

The compositional nature of the compiler is very important because it gives rise to a very general notion of foreign-function interface, expressible both as control and as communication, which allows a program to interface with other programs, in a syntax-independent way (see [13] for a discussion), opening the door to the seamless development of heterogeneous open systems in a distributed setting.

We believe we have established a solid foundational platform on which to build realistic seamless distributed compilers. Further work is needed in optimising the output of the compiler which is currently, as discussed, inefficient. The sources of inefficiency in this compiler are not just the generation of heavy-duty plumbing, but also the possibly unwise assignment of computation to nodes, requiring excessive network communication. Previous work in game semantics for resource usage can be naturally adapted to the operational setting of the GAMs and facilitate the automation of optimised task assignment [11].

References

[1] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full Abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.

[2] N. Benton. Embedded interpreters. *J. Funct. Program.*, 15(4):503–542, 2005.

[3] G. Berry and G. Boudol. The Chemical Abstract Machine. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*, pages 81–94. ACM Press, 1990.

[4] P.-L. Curien and H. Herbelin. Abstract machines for dialogue games. *CoRR*, abs/0706.2544, 2007.

[5] V. Danos, H. Herbelin, and L. Regnier. Game Semantics & Abstract Machines. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 394–405. IEEE Computer Society, 1996.

[6] V. Danos and L. Regnier. Reversible, Irreversible and Optimal lambda-Machines. *Theor. Comput. Sci.*, 227(1-2):79–97, 1999.

[7] C. Faggian and F. Maurel. Ludics Nets, a game Model of Concurrent Interaction. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 376–385. IEEE Computer Society, 2005.

[8] O. Fredriksson and D. R. Ghica. Seamless distributed computing from the geometry of interaction. In *Trustworthy Global Computing*, 2012. forthcoming.

[9] M. Gabbay and D. R. Ghica. Game Semantics in the Nominal Model. *Electr. Notes Theor. Comput. Sci.*, 286:173–189, 2012.

[10] M. Gabbay and A. M. Pitts. A New Approach to Abstract Syntax Involving Binders. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 214–224. IEEE Computer Society, 1999.

[11] D. R. Ghica. Slot games: a quantitative model of computation. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005*, pages 85–97. ACM, 2005.

[12] D. R. Ghica. Applications of Game Semantics: From Program Analysis to Hardware Synthesis. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 17–26. IEEE Computer Society, 2009.

[13] D. R. Ghica. Function interface models for hardware compilation. In *9th IEEE/ACM International Conference on Formal Methods and Models for Codesign, MEMOCODE 2011, Cambridge, UK, 11-13 July, 2011*, pages 131–142. IEEE, 2011.

[14] D. R. Ghica and A. S. Murawski. Angelic semantics of fine-grained concurrency. *Ann. Pure Appl. Logic*, 151(2-3):89–114, 2008.

[15] D. R. Ghica and N. Tzevelekos. A System-Level Game Semantics. *Electr. Notes Theor. Comput. Sci.*, 286:191–211, 2012.

[16] J.-Y. Girard. Geometry of interaction 1: Interpretation of System F. *Studies in Logic and the Foundations of Mathematics*, 127:221–260, 1989.

[17] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message passing interface*, volume 1. MIT press, 1999.

[18] J. M. E. Hyland and C.-H. L. Ong. Pi-Calculus, Dialogue Games and PCF. In *FPCA*, pages 96–107, 1995.

[19] J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.

[20] J. Laird. Exceptions, Continuations and Macro-expressiveness. In *Programming Languages and Systems, 11th European Symposium on Programming, ESOP 2002, held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings*, pages 133–146. Springer, 2002.

[21] I. Mackie. The Geometry of Interaction Machine. In *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, pages 198–208. ACM Press, 1995.

[22] R. Milner. Functions as Processes. In *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings*, pages 167–180. Springer, 1990.

[23] D. Murray, M. Schwarzkopf, C. Smowton, S. Smith, A. Madhavapeddy, and S. Hand. Ciel: a universal execution engine for distributed data-flow computing. 2011.

[24] C.-H. L. Ong. Verification of Higher-Order Computation: A Game-Semantic Approach. In *Programming Languages and Systems, 17th European Symposium on Programming, ESOP 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 299–306. Springer, 2008.