A Chart Semantics for the Pi-Calculus

Johannes Borgström[1,2]

TU Berlin

Andrew D. Gordon[3] Andrew Phillips[4]

Microsoft Research

Abstract

We present a graphical semantics for the pi-calculus, that is easier to visualize and better suited to expressing causality and temporal properties than conventional relational semantics. A pi-chart is a finite directed acyclic graph recording a computation in the pi-calculus. Each node represents a process, and each edge either represents a computation step, or a message-passing interaction. Pi-charts enjoy a natural pictorial representation, akin to message sequence charts, in which vertical edges represent control flow and horizontal edges represent data flow based on message passing. A pi-chart represents a single computation starting from its top (the nodes with no ancestors) to its bottom (the nodes with no descendants). Unlike conventional reductions or transitions, the edges in a pi-chart induce ancestry and other causal relations on processes. We give both compositional and operational definitions of pi-charts, and illustrate the additional expressivity afforded by the chart semantics via a series of examples.

Keywords: pi-calculus, causality, message sequence charts.

1 Message Sequence Charts as Process Histories

Message sequence charts (MSCs) are a successful graphical notation for describing the history of interactions between system components running in parallel. They are standardized by the ITU in connection with the Specification and Description Language (SDL) [22,21], and are included, as sequence diagrams, in the Unified Modeling Language (UML) [33]. MSCs are widely used to specify the behaviour of systems made up of multiple components; a substantial literature addresses the problems of defining formal semantics for MSCs and deriving implementation code from MSCs used as specifications [27,2].

This paper explores a different direction, the use of MSCs as a formal semantics, in terms of potential execution histories, for known code. We work within a process calculus,

[1] A long version of this paper is available at `http://lamp.epfl.ch/~jobo/chartLONG.pdf`

[2] Email: `jobo@cs.tu-berlin.de`

[3] Email: `adg@microsoft.com`

[4] Email: `aphillip@microsoft.com`

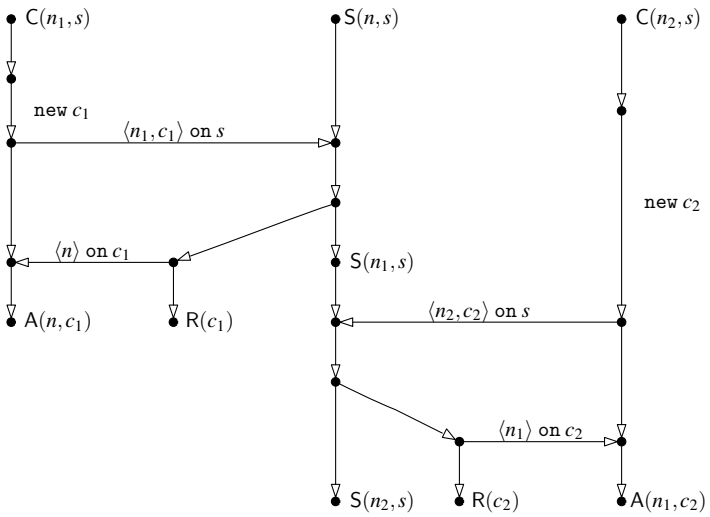
the pi-calculus, although the ideas should apply to other languages. The semantics of the pi-calculus is typically specified as a reaction or reduction relation, or as a labelled transition system [30,39]. We propose a form of MSC as an alternative.

In fact, MSCs are already used informally to illustrate computations in process calculi. For example, Bonelli and Compagnoni [5] visualize intended histories of pi-calculus processes with MSCs. Phillips, Yoshida, and Eisenbach [36] illustrate the semantics of a distributed abstract machine for the boxed ambient calculus [11] with MSCs. Jeffrey and Rathke [23] consider traces induced by a labelled transition system, and make informal connections between these and sequence diagrams. A paper [4] on the TulaFale process language uses an MSC to show an attack on a security protocol. In these papers, the formal semantics is given by relations and MSCs appear only informally. The attraction of MSCs is that they pictorially represent the identity of individual process components as they evolve and interact with other components; the conventional reduction semantics hides this information. Since the history and identity of components is valuable for expressing formal properties of systems, we go further and ask whether MSCs are suitable in themselves as a formal semantics.

To explain some of the basic ideas and to see some of the benefits of a chart semantics for the pi-calculus, we describe a simple example. We suppose there is a single stateful server $\mathsf{S}(n,s)$ which when called with a value n' and a session channel c, responds by sending on c its current state n, provisions a private service $\mathsf{R}(c)$ to handle the session, and changes state to $\mathsf{S}(n',s)$. Here is pi-calculus code for such a server, together with a client $\mathsf{C}(n',s)$ that initiates such a session, and then runs $\mathsf{A}(n,c)$ where n is the previous state of the server and c is the session channel.

$$\mathsf{S}(n,s) := s(n',c).(\overline{c}\langle n\rangle.\mathsf{R}(c) \mid \mathsf{S}(n',s))$$
$$\mathsf{C}(n',s) := (\nu c)\overline{s}\langle n',c\rangle.c(n).\mathsf{A}(n,c)$$

The pi-chart below shows interactions between one server and two clients. Pi-charts are in the spirit of MSCs but do not conform to the letter of the standard [21]. In particular, we allow processes to fork, and to generate fresh names.



A pi-chart is a directed acyclic graph. Both nodes and edges are labelled. As in this example we usually omit some node and edge labels to avoid clutter. Nodes are labelled with processes. Downward (or oblique) edges represent process evolution and are labelled with *next labels*, including `new` c, which represents the generation of a channel c, and ε, which represents the unfolding of a process constant or parallel composition. The next label ε is generally omitted. Horizontal edges represent interaction, and are labelled with *communication labels*, $\langle \widetilde{c} \rangle$ `on` a.

A pi-chart represents a single computation starting from its top (the nodes with no ancestors) to its bottom (the nodes with no descendants), with restrictions corresponding to any new names. The computation in the example corresponds to the following series of reductions in a conventional reduction semantics.

$$\mathsf{C}(n_1,s) \mid \mathsf{S}(n,s) \mid \mathsf{C}(n_2,s) \rightarrow^* (\nu c_1)(\nu c_2)(\mathsf{A}(n,c_1) \mid \mathsf{R}(c_1) \mid \mathsf{S}(n_2,s) \mid \mathsf{R}(c_2) \mid \mathsf{A}(n_1,c_2))$$

As a means of visualizing computation, pi-charts have advantages over the conventional relational semantics. A series of computation steps in the relational semantics is hard to visualize; listing the series of intermediate states can lead to an overwhelming amount of syntactic detail. Conventionally, reduction and transition relations are closed up to associativity and commutativity of parallel composition. Hence, it is hard to track the evolution of individual threads within a system. One solution is to introduce syntax for abstract locations [13], although this increases the amount of syntactic detail when visualizing reductions. On the other hand, pi-charts have a two dimensional representation that is easily rendered pictorially. The graphical structure allows detail, such as process labels, to be omitted with little risk of ambiguity. Vertical paths in a chart track the evolution of individual processes; in our example, we see that $\mathsf{S}(n,s)$ is an ancestor of $\mathsf{R}(c_1)$, $\mathsf{S}(n_2,s)$, and $\mathsf{R}(c_2)$, but not of the other processes at the bottom of the chart. (There is, though, a causal relation between $\mathsf{S}(n,s)$ and all the processes at the bottom.)

In general, MSCs have been highly successful as a means of visualising and validating dynamic behaviour of concurrent systems, and their graphical representation has also facilitated communication between groups with different backgrounds [28]. We believe that a sequence chart representation of pi-calculus computations could have similar benefits.

We proceed as follows. In Section 2 we formally define a chart semantics for a synchronous pi-calculus with mixed choice. We give three separate inductive characterizations of the set of pi-charts; Theorem 2.4 establishes the equivalence of these characterizations. As evidence for the expressivity of pi-charts, we give a series of examples of correctness properties expressible using charts. Section 3 investigates the relationship of pi-charts to a conventional reduction semantics. Theorem 3.4 shows the relation between the parallel compositions of processes at the top and bottom of a pi-chart coincides with the reflexive and transitive closure of a conventional reduction semantics, up to top-level restrictions. Theorem 3.8 relates structural congruence of processes with a structural congruence on graphs. Sections 4 and 5 conclude and discuss related work.

Appendix A shows how charts can usefully illustrate the behaviour of biological reactions expressed in the pi-calculus. Appendix B is a case study of proving properties expressible with pi-charts. We introduce a type system built from standard notions of name groups, group creation, and usage bounds on channels. Formal data flow and usage proper-

ties are conveniently expressed using charts. Theorem B.1 establishes bounds on data flow and channel usage guaranteed by the type system.

2 A Chart Semantics

We consider a polyadic pi-calculus, with synchronous communication, mixed choice, and process constants. Standard variations such as replication operators or asynchronous output can be accommodated in our framework, but we omit the details. The only unusual feature is that we annotate the autonomous τ prefixes with terms t from a free algebra $\mathbf{A}$ over names; these terms serve various purposes, such as representing events (for correspondence assertions [20]) and type annotations (for the system in Appendix B).

Syntax for Pi-Calculus Processes: P

a, c, x	names and variables
$M ::= M + M \mid \overline{a}\langle\widetilde{c}\rangle.P \mid a(\widetilde{x}).P \mid \tau_t.P$	mixed choice
$P, Q, R ::= M \mid (P \mid Q) \mid (\nu a)P \mid \mathsf{A}(\widetilde{c}) \mid \mathbf{0}$	process

Let $\mathbf{P}$ be the set of all processes. Names identify communication channels. We write $\mathrm{fn}(P)$ for the set of names occurring free in P. Let $P\{^{y}/_{x}\}$ be the outcome of substituting y for each free occurrence of x in P. We write $\widetilde{a}$, $\widetilde{c}$, $\widetilde{x}$ for finite tuples of names.

The intended meaning of the process syntax is as follows. An *output* $\overline{a}\langle\widetilde{c}\rangle.P$ sends the tuple $\widetilde{c}$ on channel a, to become P. An *input* $a(\widetilde{x}).P$ receives a tuple $\widetilde{c}$, of the same length as $\widetilde{x}$, off channel a, to become $P\{^{\widetilde{c}}/_{\widetilde{x}}\}$. In $a(\widetilde{x}).P$, the names $\widetilde{x}$ are bound with scope P, and assumed to be pair-wise distinct. A process $\tau_t.P$ autonomously marks the event t, and becomes P. A *choice* $M+N$ behaves either as M or N. A *parallel composition* $P \mid Q$ behaves as P running in parallel with Q. A *restriction* $(\nu a)P$ creates a fresh name a and becomes P; the name a is bound and has scope P. We assume a given *constant library*, a finite collection of process constants, each of which has a *definition*, written $\mathsf{A}(\widetilde{x}) := P$, where $\mathrm{fn}(P) \subseteq \widetilde{x}$. Given such a definition, a process $\mathsf{A}(\widetilde{c})$ behaves as $P\{^{\widetilde{c}}/_{\widetilde{x}}\}$. Finally, $\mathbf{0}$ does nothing.

We identify phrases of syntax up to consistent renaming of bound names; for instance, $(\nu a)P = (\nu b)P\{^{b}/_{a}\}$ if $b \notin \mathrm{fn}(P)$. We also identify processes up to associativity and commutativity of the choice operator.

2.1 *Labelled graphs*

Charts are particular labelled graphs. Nodes are drawn from an infinite set of *node identifiers*, $\mathbf{I}$, ranged over by ι. Nodes are labelled with pi-calculus processes. Each edge has either a *next label* ($n\ell$) or a *communication label* ($\langle\widetilde{c}\rangle$ `on` a). A next label represents an event, and labels an edge from a process to its successor; the next label `new` x, where the free name x is globally fresh, represents name generation. Annotation t represents a tau step, while next step ε represents all other kinds of process evolution, including unfolding of process constants and parallel compositions. A communication label represents a message passing from an output to an input.

Edge Labels for the Pi-Calculus: nL and L

$n\ell \in \mathbf{nL} ::=$		next label
	$\texttt{new}\, x$	name generation
	t	annotation
	ε	next step
$\ell \in \mathbf{L} ::=$		edge label
	$n\ell$	next label
	$\langle \widetilde{c} \rangle \,\texttt{on}\, a$	communication

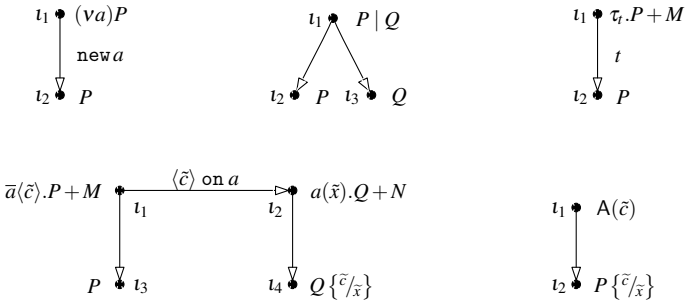
A labelled graph is a pair (N,E) where $\mathsf{N} : \mathbf{I} \to \mathbf{P}$ and $\mathsf{E} : \mathbf{I} \times \mathbf{I} \to \mathbf{L}$ are finite maps. Given $G = (\mathsf{N},\mathsf{E})$, we write N_G for N and E_G for E. A graph G is *well-formed* iff $\mathrm{dom}(\mathsf{E}_G) \subseteq (\mathrm{dom}(\mathsf{N}_G) \times \mathrm{dom}(\mathsf{N}_G))$. The following notations express graphs as compositions of labelled nodes and edges.

$$\begin{aligned} \iota \overset{\ell}{\rightarrowtriangle} \iota' &:= (\varnothing, \{((\iota,\iota'),\ell)\}) \\ \iota \bullet P &:= (\{(\iota,P)\},\varnothing) \\ G \cup H &:= (\mathsf{N}_G \cup \mathsf{N}_H, \mathsf{E}_G \cup \mathsf{E}_H) \text{ when a well-formed graph} \\ G \setminus H &:= (\mathsf{N}_G \setminus \mathsf{N}_H, \mathsf{E}_G \setminus \mathsf{E}_H) \end{aligned}$$

2.2 *Primitive pi-charts*

We begin our chart semantics by defining a set of primitive pi-charts. Let a *primitive chart* be any instance of one of the following five schemas. Here and elsewhere we omit the ε label from edges. We refer to nodes with the variables $\iota_1, \iota_2, \iota_3, \iota_4$, assumed pair-wise distinct. Let $\mathbf{C}_p$ be the set of primitive charts.

Primitive Charts: $\mathbf{C}_p$



when $\mathsf{A}(\widetilde{x}) := P$

Since we identify processes up to renaming of bound names, from $(\nu a)P$ we get infinitely many primitive charts of the first form above, one for each possible choice of a.

2.3 *The top and bottom of pi-charts*

Each pi-chart has a *top*, the nodes with no predecessors, and a *bottom*, the nodes with no successors. A core idea, formalized later as Theorem 3.4, is that a pi-chart represents a computation starting with the processes at the top, and ending with those at the bottom. We

formalize top and bottom below, together with other notations needed for a compositional definition of pi-charts: $\mathsf{new}(G)$ is the set of names generated within a chart; G_{nil} is the edgeless graph consisting of the terminal nodes of G, that is, those labelled with $\mathbf{0}$.

$$\begin{aligned} G_\top &:= (\{(\iota, P) \mid \mathsf{N}_G(\iota) = P \wedge \neg(\exists \iota', n\ell.\ \mathsf{E}_G(\iota', \iota) = n\ell)\}, \varnothing) \\ G_\bot &:= (\{(\iota, P) \mid \mathsf{N}_G(\iota) = P \wedge \neg(\exists \iota', n\ell.\ \mathsf{E}_G(\iota, \iota') = n\ell)\}, \varnothing) \\ \mathsf{new}(G) &:= \{a \mid \mathtt{new}\ a \in \mathrm{range}(\mathsf{E}_G)\} \\ G_{\mathsf{nil}} &:= (\{(\iota, \mathbf{0}) \mid \mathsf{N}_G(\iota) = \mathbf{0}\}, \varnothing) \\ \mathsf{I}_G &:= \mathrm{dom}(\mathsf{N}_G) \end{aligned}$$

We write $\mathsf{nn}_G(S)$ for $\cup_{\iota \in S} \mathrm{fn}(\mathsf{N}_G(\iota))$ when $S \subseteq \mathsf{I}_G$. When speaking about a particular graph G, we often write $\mathsf{nn}(S)$ for $\mathsf{nn}_G(S)$. We let $\mathsf{nn}(G) := \mathsf{nn}_G(\mathsf{I}_G)$. One invariant we want to preserve is that all names that occur in a chart are either free in the processes at the top of the chart or freshly created. A *well-named* chart is one satisfying $\mathsf{nn}(G) \subseteq \mathsf{nn}(G_\top) \uplus \mathsf{new}(G)$; note that all primitive charts are well-named.

2.4 Three equivalent characterizations of pi-charts

We can now define how to build larger charts from primitive ones. We give three definitions, two compositional and one operational in flavour, and show them equivalent.

Intuitively, two pi-charts may be composed in sequence if the bottom of the first equals the top of the second. Dually, two pi-charts may be composed in parallel if they are completely disjoint. Given these notions, a *pi-chart* is either a singleton chart $\iota \bullet P$, a primitive chart $G \in \mathbf{C}_p$, or a composition $G \cup H$ where G and H are composable, either in sequence or in parallel.

The following definitions make these intuitions precise; various freshness conditions are needed to guarantee global uniqueness of generated names.

Sequential Composition: $S(G,H)$

If G and H are well-formed then $S(G,H)$ iff
(1) $\mathsf{I}_G \cap \mathsf{I}_H = \mathsf{I}_{G_\bot} \setminus \mathsf{I}_{(G_\bot)_{\mathsf{nil}}} = \mathsf{I}_{H_\top}$;
(2) $\mathsf{new}(H) \cap \mathsf{new}(G) = \mathsf{nn}(G_\top) \cap \mathsf{new}(H) = \varnothing$; and
(3) whenever $\iota \in \mathsf{I}_G \cap \mathsf{I}_H$ then $\mathsf{N}_G(\iota) = \mathsf{N}_H(\iota)$.

Parallel Composition: $P(G,H)$

If G and H are well-formed then $P(G,H)$ iff
(1) $\mathsf{I}_G \cap \mathsf{I}_H = \varnothing$; and
(2) $\mathsf{new}(G) \cap \mathsf{new}(H) = \mathsf{nn}(G_\top) \cap \mathsf{new}(H) = \mathsf{nn}(H_\top) \cap \mathsf{new}(G) = \varnothing$.

A First Characterization of Pi-Charts: $\mathbf{C}_{SP}$

$$\frac{}{\iota \bullet P \in \mathbf{C}_{SP}} \qquad \frac{G \in \mathbf{C}_p}{G \in \mathbf{C}_{SP}} \qquad \frac{G,H \in \mathbf{C}_{SP} \quad S(G,H)}{G \cup H \in \mathbf{C}_{SP}} \qquad \frac{G,H \in \mathbf{C}_{SP} \quad P(G,H)}{G \cup H \in \mathbf{C}_{SP}}$$

Although sequential and parallel compositions are intuitive and easy to define, they lack some algebraic properties useful in proofs. As an example, if $P(G_1, G_2)$ and $S(G_1 \cup G_2, H)$, we neither have $S(G_1, H)$ nor $P(G_1, H)$, in general. Moreover, inductive proofs using the definition of $\mathbf{C}_{SP}$ require two inductive cases, where one ought to suffice. To overcome these problems, we unify parallel and sequential composition into *liberal composition*, and obtain a second definition of pi-charts.

Liberal Composition: $L(G,H)$

If G and H are well-formed then $L(G,H)$ ("G before H") iff
(1) $\mathsf{I}_G \cap \mathsf{I}_H \subseteq \mathsf{I}_{G_\perp}$ and $\mathsf{I}_G \cap \mathsf{I}_H \subseteq \mathsf{I}_{H_\top}$;
(2) $\mathsf{new}(H) \cap \mathsf{new}(G) = \mathsf{nn}(G_\top) \cap \mathsf{new}(H) = \mathsf{nn}(H_\top \setminus G) \cap \mathsf{new}(G) = \varnothing$; and
(3) whenever $\iota \in \mathsf{I}_G \cap \mathsf{I}_H$ then $\mathsf{N}_G(\iota) = \mathsf{N}_H(\iota)$.

A Second Characterization of Pi-Charts: $\mathbf{C}_L$

$$\frac{}{\iota \bullet P \in \mathbf{C}_L} \qquad \frac{G \in \mathbf{C}_p}{G \in \mathbf{C}_L} \qquad \frac{G \in \mathbf{C}_L \quad H \in \mathbf{C}_L \quad L(G,H)}{G \cup H \in \mathbf{C}_L}$$

By comparing definitions, it is clear that liberal composition is more permissive than either parallel or sequential composition. Crucially, liberal composition is associative, and preserves well-namedness.

Lemma 2.1 *Assume that graphs G_1, G_2, G_3 are well-named.*

(1) *If $L(G_1, G_2)$ and $L(G_1 \cup G_2, G_3)$, then $L(G_2, G_3)$ and $L(G_1, G_2 \cup G_3)$.*

(2) *If $L(G_2, G_3)$ and $L(G_1, G_2 \cup G_3)$, then $L(G_1, G_2)$ and $L(G_1 \cup G_2, G_3)$.*

Lemma 2.2 *If G_1, G_2 are well-named and $L(G_1, G_2)$ then $G_1 \cup G_2$ is well-named.*

Proof

$$\begin{aligned}
&n(G_1 \cup G_2) \\
=&n(G_1) \cup n(G_2) \\
\subseteq&\mathsf{nn}(G_{1\top}) \cup \mathsf{new}(G_1) \cup \mathsf{nn}(G_{2\top}) \cup \mathsf{new}(G_2) \\
=&\mathsf{new}(G_1 \cup G_2) \cup \mathsf{nn}(G_{1\top}) \cup n(\mathsf{I}_{G_{2\top}}) \\
\subseteq&\mathsf{new}(G_1 \cup G_2) \cup \mathsf{nn}(G_{1\top}) \cup n(\mathsf{I}_{G_{2\top}} - \mathrm{join}(G_1, G_2)) \cup n(\mathrm{join}(G_1, G_2)) \\
=&\mathsf{new}(G_1 \cup G_2) \cup \mathsf{nn}(G_1 \cup G_{2\top}) \cup n(\mathrm{join}(G_1, G_2))
\end{aligned}$$

Then $n(\mathrm{join}(G_1, G_2)) \subseteq n(\mathsf{I}_{G_{1\perp}}) \subseteq n(G_1) \subseteq \mathsf{nn}(G_{1\top}) \cup \mathsf{new}(G_1) \subseteq n(\mathsf{I}_{G_{1\top}} \cup (\mathsf{I}_{G_{2\top}} \setminus \mathsf{I}_{G_1})) \cup \mathsf{new}(G_1) \subseteq \mathsf{nn}(G_1 \cup G_{2\top}) \cup \mathsf{new}(G_1 \cup G_2)$.

Moreover,

$$\begin{aligned}
&\mathsf{nn}(G_1 \cup G_{2\top}) \cap \mathsf{new}(G_1 \cup G_2) \\
=&(\mathsf{nn}(G_{1\top}) \cup \mathsf{nn}(\mathsf{I}_{G_{2\top}} - \mathsf{I}_{G_{1\perp}})) \cap (\mathsf{new}(G_1) \cup \mathsf{new}(G_2)) \\
=&(\mathsf{new}(G_1) \cap \mathsf{nn}(\mathsf{I}_{G_{2\top}} - \mathsf{I}_{G_{1\perp}})) \\
=&\varnothing
\end{aligned}$$

□

1 *If $G \in \mathbf{C}_L$ then G is well-named.*

Proof By Lemma 2.2 and $\mathsf{nn}(G) \subseteq \mathsf{nn}(G_\top) \cup \mathsf{new}(G)$ for all primitive pi-charts G.

By associativity (Lemma 2.1) we obtain the following iterative account of $\mathbf{C}_L$.

Lemma 2.3 *$G \in \mathbf{C}_L$ iff there exist pi-charts $H_1, \ldots, H_n \in \mathbf{C}_p \cup \{\iota \bullet P \mid \iota \in \mathbf{I}, P \in \mathbf{P}\}$ such that $G = H_1 \cup \ldots \cup H_n$ and $L(H_1 \cup \cdots \cup H_{i-1}, H_i)$ for each $i \in 2..n$.*

Proof The implication from right to left holds by a simple induction on n.

We prove the other direction for all derivations D of $G \in \mathbf{C}_L$ by induction on the depth of derivations of $G \in \mathbf{C}_L$. We define the right complexity r of a derivation D as $r(D) := 0$ for a single primitive rule D and $r(\text{B-BEFORE}(D_1, D_2)) := r(D_1) + (r(D_2) + 1)^2$.

The base case ($G \in \mathbf{C}_P \cup \{\iota \bullet P \mid \iota \in \mathbf{I}, P \in \mathbf{P}\}$) is trivial. For the induction case, we assume that the lemma holds for all pi-charts $G \in \mathbf{C}_L$ with derivations of depth strictly less than k.

Assume that $D := \text{B-BEFORE}(D_1, D_2)$ is a derivation for $G \in \mathbf{C}_L$ with depth k and the smallest $r(D)$ among all derivations for G with depth k, and that D_1 and D_2 are derivations of $G_1 \in \mathbf{C}_L$ and $G_2 \in \mathbf{C}_L$, respectively. If D_2 is a leaf then $G_2 = H \in \mathbf{C}_p \cup \{\iota \bullet P \mid \iota \in \mathbf{I}, P \in \mathbf{P}\}$, and we have by the induction hypothesis that $G_1 = H_1 \cup \ldots \cup H_n$ where $H_1, \ldots, H_n$ are in $\mathbf{C}_p \cup \{\iota \bullet P \mid \iota \in \mathbf{I}, P \in \mathbf{P}\}$ such that $L(H_1 \cup \cdots \cup H_{i-1}, H_i)$ for each $i \in 2..n$. Thus, $G = H_1 \cup \ldots \cup H_n \cup H$, and moreover $L(G_1, H)$ by assumption.

If D_2 is not a leaf, we derive a contradiction. In this case, $D_2 = \text{B-BEFORE}(D_{21}, D_{22})$ for some D_{21}, D_{22} deriving G_{21} and G_{22}, respectively. Lemma 2.1 then gives that $L(G_1, G_{21})$ and $L(G_1 \cup G_{21}, G_{22})$. Then $\text{B-BEFORE}(\text{B-BEFORE}(D_1, D_{21}), D_{22})$ is a derivation of $G \in \mathbf{C}_L$ and

$$\begin{aligned} & r(\text{B-BEFORE}(\text{B-BEFORE}(D_1, D_{21}), D_{22})) \\ &= r(D_1) + (r(D_{21}) + 1)^2 + (r(D_{22}) + 1)^2 \\ &< r(D_1) + (r(D_{21}) + 1 + (r(D_{22}) + 1)^2)^2 \\ &= r(D) \end{aligned}$$

which is a contradiction.

For our final definition, we start with an initial set of unconnected nodes and add primitive charts one by one to the bottom. This amounts to an operational semantics. (We use it as the basis of two separate pi-calculus implementations that output pi-charts in the dot language, suitable for rendering with Graphviz [19].) We define *chart extension* $G \to G'$ ("G extends to G'") as follows, and hence obtain a third characterization of pi-charts.

Chart Extension $G \to G'$ and a Third Characterization of Pi-Charts $\mathbf{C}_I$

$G \to G'$ iff there is $H \in \mathbf{C}_p$ such that $G' = G \cup H$ and $L(G, H)$ and $\mathsf{I}_{H_\top} \subseteq \mathsf{I}_{G_\bot}$
$\mathbf{C}_I := \{G \mid G_\top \to^* G\}$

Theorem 2.4 $\mathbf{C}_{SP} = \mathbf{C}_L = \mathbf{C}_I$

Proof We begin by proving that $\mathbf{C}_L \subseteq \mathbf{C}_I$, that is, that $G \in \mathbf{C}_L$ implies that $G_\top \to^* G$.

Trivially, $L(G_\top, G)$. By Lemma 2.3, there exist primitive pi-charts $H_1, \ldots, H_n$ such that $G = H_1 \cup \ldots \cup H_n$ and $L(H_1 \cup \cdots \cup H_{i-1}, H_i)$ for each $i \in 1..n$. Since $L(G_\top, H_1 \cup \ldots \cup H_n)$ Lemma 2.1(i) gives that $L(G_\top, H_1 \cup \ldots \cup H_{n-1})$ and $L(G_\top \cup H_1 \cup \ldots \cup H_{n-1}, H_n)$.

We also have $\mathsf{I}_{H_{n\top}} \subseteq \mathsf{I}_{(G_\top \cup H_1 \cup \ldots \cup H_{n-1})_\bot}$, so $G_\top \cup H_1 \cup \ldots \cup H_{n-1} \to G$. Inductively, $G_\top \to^n G$.

Secondly, we prove that $\mathbf{C}_I \subseteq \mathbf{C}_{SP}$, that is, that if $G_\top \to^* G$ then $G \in \mathbf{C}_{SP}$, by induction on the number of extensions. For the base case, $G = G_\top = (\mathsf{N}_G, \varnothing) \in \mathbf{C}_{SP}$, by parallel composition of charts of the form $\iota \bullet P$. For the induction case we have $G = G' \cup H$ with $G' \in \mathbf{C}_I$, $H \in \mathbf{C}_p$, $L(G', H)$ and $\mathsf{I}_{H_\top} \subseteq \mathsf{I}_{G'_\bot}$. As above $(\mathsf{I}_{G'_\bot} \setminus \mathsf{I}_{G'_{\mathsf{nil}}}) \cup H \in \mathbf{C}_{SP}$. By induction $G' \in \mathbf{C}_{SP}$. Since $\mathbf{0} \notin \mathrm{dom}(\mathsf{N}_{\mathsf{I}_{H_\top}})$ we get $S(G', (\mathsf{I}_{G_\bot} \setminus \mathsf{I}_{G'_{\mathsf{nil}}}) \cup H)$, so $G' \cup H \in \mathbf{C}_{SP}$.

Finally, since $S(G,H) \vee P(G,H) \Longrightarrow L(G,H)$, $\mathbf{C}_{SP} \subseteq \mathbf{C}_L$ by induction. □

2.5 *Expressible properties*

To end this section, we discuss some properties expressible with pi-charts. We may see the edges of a chart G as a relation $\twoheadrightarrow_G \subseteq \mathbf{I} \times \mathbf{L} \times \mathbf{I}$. We split this relation, writing $\iota_1 \overset{\mathtt{on}}{\twoheadrightarrow}_G \iota_2$ for $\exists \widetilde{c}, a.\ \iota_1 \xrightarrow{\langle \widetilde{c} \rangle\ \mathtt{on}\ a}_G \iota_2$ and $\iota_1 \downarrow\!\!\!\downarrow \iota_2$ for $\mathsf{E}_G(\iota_1, \iota_2) \in \mathbf{nL}$. Hence, we define some causal relations, roughly following the terminology of Priami [38].

Causal Relations:

$$[\text{Ancestor}]_G := \downarrow\!\!\!\downarrow_G^* \qquad [\text{Causes}]_G := (\downarrow\!\!\!\downarrow_G \cup \overset{\mathtt{on}}{\twoheadrightarrow}_G)^* \qquad [\text{Enables}]_G := (\downarrow\!\!\!\downarrow_G \cup \overset{\mathtt{on}}{\twoheadrightarrow}_G \cup (\overset{\mathtt{on}}{\twoheadrightarrow}_G)^{-1})^*$$

The node receiving a message enables the sending node and all of its descendants. This is due to the synchronous nature of communication: the sender proceeds with the knowledge that the message was received, just as if they had received an explicit acknowledgement of reception. The "causes" relation only flows in the direction of communicated messages; it is the equivalent in our setting to Lamport's "happened before" relation [26].

Another causal semantics for the pi-calculus is proved semantics [9,17,16], which makes a distinction between *subject* and *object* dependencies [7]. Since the latter are only defined in terms of "bound output" labels of a labelled transition system, they have no direct counterpart in our setting where all communication is internal to a pi-chart.

Let the nodes with ι as an ancestor be the *descendants* of ι. If ι_2, ι_3 are the nodes in the primitive chart for parallel composition, the sets of descendants of ι_2 and ι_3 are disjoint. (The "causes" and "enables" relations do not possess this property.)

Lemma 2.5 *If a pi-chart G has distinct edges $\iota_1 \overset{\varepsilon}{\twoheadrightarrow} \iota_2$ and $\iota_1 \overset{\varepsilon}{\twoheadrightarrow} \iota_3$ then there is no ι_4 such that both ι_2 $[Ancestor]_G$ ι_4 and ι_3 $[Ancestor]_G$ ι_4.*

We can concisely express some intensional properties of the interactions recorded in a chart G as follows (omitting the subscripts $_G$).

- "I got an answer to this message ($\iota_1 \xrightarrow{\langle a \rangle\ \mathtt{on}\ \widetilde{c}} \iota_2$)."

$$\exists \iota'.\ (\iota_1\ [\text{Ancestor}]\ \iota') \wedge (\iota_2\ [\text{Ancestor}] \overset{\mathtt{on}}{\twoheadrightarrow} \iota')$$

- "Every $end(t)$ event was caused by a corresponding $begin(t)$ event." [20]

$$\forall t, \iota_1, \iota_2 \exists \iota'.\ (\iota_1 \xrightarrow{end(t)} \iota_2) \Longrightarrow (\iota' \xrightarrow{begin(t)} [\text{Causes}]\ \iota_1)$$

- "I (ι_1) only communicated with descendants of somebody else (ι_2)."

$$\forall \iota' (\iota_1\ [\text{Ancestor}] \xrightarrow{\texttt{on}} \iota') \Longrightarrow (\iota_2\ [\text{Ancestor}]\ \iota')$$

- "No name created by me (ι_1) was ever transmitted to somebody else (ι_2)."

$$\neg\exists \iota_1', \iota_2', a, \widetilde{c}.\ (b \in \widetilde{c}) \wedge (\iota_1\ [\text{Ancestor}] \xrightarrow{\texttt{new}\ b} \iota_1') \wedge (\iota_1'\ [\text{Ancestor}] \xrightarrow{\langle \widetilde{c} \rangle\ \texttt{on}\ a} \iota_2') \wedge (\iota_2\ [\text{Ancestor}]\ \iota_2')$$

3 Relating the Reduction Semantics and Chart Semantics

We define a standard reduction semantics for our pi-calculus [30,39], based on structural equivalence $P \equiv Q$, and reduction $P \to Q$. The only noteworthy detail is that constant instantiation is a rule of reduction, not structural equivalence. This avoids the syntactic constraints on definitions usually needed to avoid any unbounded unfolding.

Structural Equivalence: $P \equiv Q$

$P \equiv P$	STRUCT-REFL
$Q \equiv P \Rightarrow P \equiv Q$	STRUCT-SYMM
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	STRUCT-TRANS
$P \equiv P' \Rightarrow (\nu x)P \equiv (\nu x)P'$	STRUCT-RES
$P \equiv P' \Rightarrow P \mid Q \equiv P' \mid Q$	STRUCT-PAR
$P \mid Q \equiv Q \mid P$	STRUCT-PAR-COMM
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	STRUCT-PAR-ASSOC
$a \notin \text{fn}(P) \Rightarrow (\nu a)(P \mid Q) \equiv P \mid (\nu a)Q$	STRUCT-RES-PAR
$(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$	STRUCT-RES-RES

Reduction: $P \to Q$

$P \to P' \Rightarrow P \mid Q \to P' \mid Q$	RED-PAR
$P \to P' \Rightarrow (\nu a)P \to (\nu a)P'$	RED-RES
$P \equiv Q, Q \to Q', Q' \equiv P' \Rightarrow P \to P'$	RED-STRUCT
$(a(\widetilde{x}).P + M) \mid (\overline{a}\langle \widetilde{c} \rangle.Q + N) \to P\left\{{}^{\widetilde{c}}/_{\widetilde{x}}\right\} \mid Q$	RED-COMM
$\tau_t.P + M \to P$	RED-NOTE
$\mathsf{A}(\widetilde{c}) \to P\left\{{}^{\widetilde{c}}/_{\widetilde{x}}\right\}$ if $\mathsf{A}(\widetilde{x}) := P$	RED-INST

3.1 Operational correspondence

We now develop the correspondence between the reduction semantics of the pi-calculus and pi-charts. We begin by defining the process corresponding to a pi-chart: the parallel composition of the processes at the bottom of the chart inside a restriction of the names generated in the chart.

Unloading a pi-chart G to a process: $[\![G]\!]$

$[\![G]\!] := (\nu \mathsf{new}(G))(\prod_{\iota \in \mathsf{I}_{G_\perp}} \mathsf{N}_G(\iota))$ (hence: $[\![G]\!] = (\nu \mathsf{new}(G))[\![G_\perp]\!]$)

1 *Given P, we let $N(P)$ be the set of processes of the form $(\nu \widetilde{e})(P_1 \mid \cdots \mid P_{n_1} \mid \mathsf{A}_1(\widetilde{c}_1) \mid \cdots \mid \mathsf{A}_{n_2}(\widetilde{c}_{n_2}) \mid \mathbf{0}_{t_1} \mid \cdots \mid \mathbf{0}_{t_{n_2}})$, where each P_i is a sum M_i or a constant $\mathsf{A}_i(\widetilde{c}_i)$, that are structurally equivalent to P. We write P_N for any element of $N(P)$.*

We split the primitive charts into housekeeping charts, that do not correspond to reduction steps, and computation charts, that do. Let the set of *housekeeping charts*, $\mathbf{C}_h$, be the subset of $\mathbf{C}_p$ generated just from the schemas for parallel composition and restriction. Let the set of *computation charts*, $\mathbf{C}_c$, be $\mathbf{C}_p \setminus \mathbf{C}_h$. Similarly, we split the chart extension relation $\rightarrow$ into two relations $\rightarrow_h$ and $\rightarrow_c$ as follows. If $G \rightarrow G \cup H$ with $H \in \mathbf{C}_h$, we write $G \rightarrow_h G \cup H$. Similarly $G \rightarrow_c G \cup H$ if $G \rightarrow G \cup H$ with $H \in \mathbf{C}_c$.

Housekeeping charts

We can then say that a pi-chart G is in normal form if $G \not\rightarrow_h$. We let $N(G)$ be the set of pi-charts G_N in normal form such that $G \rightarrow_h^* G_N$.

Lemma 3.1 *$N(G)$ is non-empty for every pi-chart G.*

Proof By induction on $\Sigma_{\iota \in \mathsf{I}_{G_\perp}} |\mathsf{N}_G(\iota)|$, where $|P|$ denotes the syntactic size of P. There are three cases.

(1) If $G_\perp = \iota \bullet (\nu a)P \cup H$ with $a \notin \mathsf{nn}(G_\top) \cup \mathsf{new}(G)$ then there is ι' with $G \rightarrow_h G \cup \mathsf{New}\langle \iota \rangle (a,P)\langle \iota' \rangle =: G'$ and ${G'}_\perp = \iota' \bullet P \cup H$. Then $\Sigma_{\iota \in \mathsf{I}_{G_\perp}} |\mathsf{N}_G(\iota)| = 1 + \Sigma_{\iota \in \mathsf{I}_{{G'}_\perp}} |\mathsf{N}_{G'}(\iota)|$, and by induction $G' \rightarrow_h^* G_N$ with G_N in normal form.

(2) Otherwise, if $G_\perp = \iota \bullet P \mid Q \cup H$ with $a \notin \mathsf{nn}(G_\top) \cup \mathsf{new}(G)$ then there is ι_1, ι_2 with $G \rightarrow_h G \cup \mathsf{Par}\langle \iota \rangle (P,Q)\langle \iota_1, \iota_2 \rangle =: G'$ and ${G'}_\perp = \{\bullet iota_1 P \cup \iota_2 \bullet Q \cup H$. Then $\Sigma_{\iota \in \mathsf{I}_{G_\perp}} |\mathsf{N}_G(\iota)| = 1 + \Sigma_{\iota \in \mathsf{I}_{{G'}_\perp}} |\mathsf{N}_{G'}(\iota)|$ and by induction $G' \rightarrow_h^* G_N$ with G_N in normal form.

(3) Otherwise, G is in normal form and we are done.

We can then show that housekeeping extension does not change the process corresponding to the chart, up to structural equivalence.

Lemma 3.2 *Suppose G is a pi-chart and $P \equiv [\![G]\!]$. If $G \rightarrow_h G \cup H$ then $P \equiv [\![G \cup H]\!]$.*

Proof If H is a PAR chart, this is immediate by AC of parallel composition. If H is a NEW chart, there is $\iota \in \mathsf{I}_{G_\perp}$ such that $\mathsf{N}_G(\iota) = (\nu x)P$, and $y \notin \mathsf{new}(G) \cup \mathsf{nn}(G_\top) \cup \mathsf{fn}((\nu x)P)$. By Corollary 1, $y \notin \mathsf{fn}(\prod_{\iota_i \in \mathsf{I}_{G_\perp}} \mathsf{N}_G(\iota_i))$, so we may use scope extrusion to conclude that $[\![G]\!] \equiv [\![G']\!]$. By transitivity, $P \equiv [\![G']\!]$.

2 *If $P \equiv [\![G]\!]$ and $G_N \in N(G)$ then $[\![G_N]\!] \in N(P)$.*

Proof For each $G_N \in N(G)$, $[\![G_N]\!] \in N([\![G_N]\!]) = N([\![G]\!]) = N(P)$.

3 *$N(P)$ is non-empty for every process P.*

Proof By Corollary 2 and Lemma 3.1 with $G = [\![\iota \bullet P]\!]$.

Reductions, on the other hand, are matched one for one by computation extension of charts, possibly with some housekeeping beforehand to reveal the redex.

Lemma 3.3 *Suppose G is a pi-chart and $P \equiv [\![G]\!]$.*

(1) *If $P \to P'$ then $G \to_h^* \to_c G'$ with $[\![G']\!] \equiv P'$.*

(2) *If $G \to_c G'$ then $P \to P'$ with $[\![G']\!] \equiv P'$.*

Proof

(1) Assume that $P_N = (\nu \widetilde{e}) \prod_{v=1\ldots n} P_v$. There are three base cases for $P \to P'$.

RED COMM Then $P \to P'$ if and only if there are $i, j \leq n$ and $a, \widetilde{c}, \widetilde{x}, P_i', P_j', M, N$ with $P_i = \overline{a}\langle b \rangle.P_i' + M$ and $P_j = a(x).P_j' + N$ such that $P' \equiv (\nu \widetilde{e})(P_i' \mid P_j' \{^{\widetilde{c}}/_{\widetilde{x}}\} \mid \prod_{v \in (\{1\ldots n\} \setminus \{i,j\})} P_v) =: Q$. By STRUCT, $P_N \to P'$.

By Corollary 2, $G \to_h^* G_N$ with $P_N =_\alpha [\![G_N]\!]$. Thus, there are $\iota_1, \iota_2 \in \mathsf{I}_{G_{N\perp}}$ with $\mathsf{N}_{G_N}(\iota_1) = \overline{a}\langle \widetilde{c} \rangle.P_i' + M$ and $\mathsf{N}_{G_N}(\iota_2) = a(\widetilde{x}).P_j' + N$. Choosing distinct $\iota_1', \iota_2' \notin \mathsf{I}_{G_N}$, we get $G_N \to_c G_N \cup \mathsf{Comm}\langle \iota_1, \iota_2 \rangle (a, \widetilde{c}, \widetilde{x}, P_i', P_j', M, N) \langle \iota_1', \iota_2' \rangle =: H$. Clearly, $[\![H]\!] \equiv Q \equiv P'$.

RED NOTE Then $P \to P'$ if and only if there is $i \leq n$ and T, P_i' with $P_i = \tau_t.P_i'$ such that $P' \equiv (\nu \widetilde{e})(\mathbf{0}_t \mid P_i' \mid \prod_{v \in (\{1\ldots n\} \setminus \{i,j\})} P_v) =: Q$. By STRUCT, $P_N \to P'$.

By Corollary 2, $G \to_h^* G_N$ with $P_N =_\alpha [\![G_N]\!]$. Thus, there is $\iota \in \mathsf{I}_{G_{N\perp}}$ with $\mathsf{N}_{G_N}(\iota) = \tau_t.P_i'$. Choosing $\iota' \notin \mathsf{I}_{G_N}$, we get $G_N \to_c G_N \cup \mathsf{Note}\langle \iota \rangle (t, P) \langle \iota' \rangle$. Clearly, $[\![G_N \cup \mathsf{Note}\langle \iota \rangle (t, P) \langle \iota' \rangle]\!] \equiv Q \equiv P'$.

RED INST Then $P \to P'$ if and only if there is $i \leq n$ and $\mathsf{A}, \widetilde{c}$ with $P_i = \mathsf{A}(\widetilde{c})$ and $\mathsf{A}(\widetilde{x}) := P_\mathsf{A}$ such that $P' \equiv (\nu \widetilde{e})(P_\mathsf{A} \{^{\widetilde{c}}/_{\widetilde{x}}\} \mid \prod_{v \in (\{1\ldots n\} \setminus \{i,j\})} P_v) =: Q$. By STRUCT, $P_N \to P'$.

By Corollary 2, $G \to_h^* G_N$ with $P_N =_\alpha [\![G_N]\!]$. Thus, there is $\iota \in \mathsf{I}_{G_{N\perp}}$ with $\mathsf{N}_{G_N}(\iota) = \mathsf{A}(\widetilde{c})$. Choosing $\iota' \notin \mathsf{I}_{G_N}$, we get $G_N \to_c G_N \cup \mathsf{Inst}\langle \iota \rangle (\mathsf{A}, \widetilde{c}) \langle \iota' \rangle$. Clearly, $[\![G_N \cup \mathsf{Inst}\langle \iota \rangle (\mathsf{A}, \widetilde{c}) \langle \iota' \rangle]\!] \equiv Q \equiv P'$.

(2) Assume that $G \to_c G \cup H$, $\mathsf{I}_{G_\perp} = \{\iota_i\}_{i=1}^n$ and $\mathsf{N}_G(\iota_i) = P_i$. Modulo renumbering of the node identifiers, there are three cases for H.

$H = \mathsf{Note}\langle \iota_1 \rangle (T, P_1') \langle \iota_1' \rangle$**:** We then have $[\![G]\!] \equiv (\nu \widetilde{e})(\tau_T.P_1' + M \mid \prod_{v=2\ldots n} P_v) \to (\nu \widetilde{e})(\mathbf{0}_T \mid P_1' \mid \prod_{v=2\ldots n} P_v) \equiv [\![G']\!]$.

$H = \mathsf{Inst}\langle \iota_1 \rangle (\mathsf{A}, \widetilde{c}) \langle \iota_1' \rangle$**:** Assuming that $\mathsf{A}(\widetilde{x}) := P_\mathsf{A}$, we then have $[\![G]\!] \equiv (\nu \widetilde{e})(\mathsf{A}(\widetilde{c}) \mid \prod_{v=2\ldots n} P_v) \to (\nu \widetilde{e})(P_\mathsf{A} \{^{\widetilde{c}}/_{\widetilde{x}}\} \mid \prod_{v=2\ldots n} P_v) \equiv [\![G']\!]$.

$H = \mathsf{Comm}\langle \iota_1, \iota_2 \rangle (a, \widetilde{c}, \widetilde{x}, P_1', P_2', M, N) \langle \iota_1', \iota_2' \rangle$**:** We then have $[\![G]\!] \equiv (\nu \widetilde{e})(\overline{a}\langle \widetilde{c} \rangle.P_1' + M \mid a(\widetilde{x}).P_2' + N \mid \prod_{v=3\ldots n} P_v) \to (\nu \widetilde{e})(P_1' \mid P_2' \{^{\widetilde{c}}/_{\widetilde{x}}\} \mid \prod_{v=3\ldots n} P_v) \equiv [\![G']\!]$.

The full correspondence between many-step reduction of processes and pi-charts is then given by the following theorem.

Theorem 3.4 *$P \to^* Q$ iff there is a pi-chart G with $P \equiv [\![G_\top]\!]$ and $Q \equiv (\nu \mathsf{new}(G))[\![G_\perp]\!]$.*

Proof By Lemma 3.2, Lemma 3.3 and induction, with G and G' in the lemmas given by $G := G_\top$ and $G' := G$. □

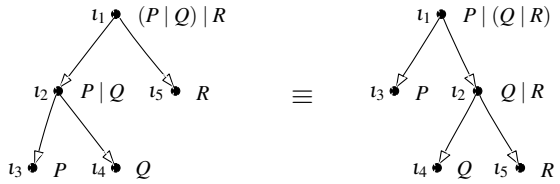
Conversely, if $[\![G_\top]\!] \to^* (\nu \mathsf{new}(G))[\![G_\bot]\!]$ for some graph G, the graph is not necessarily a pi-chart. It may have spurious edges, for example. We cannot expect to recover the notion of a pi-chart simply from the reduction semantics.

Many standard equivalences, such as barbed equivalence and congruence, are defined in terms of the relation $P \to^* Q$, plus direct observations of process structure [39]. Theorem 3.4 provides a basis for re-defining such equivalences in terms of charts.

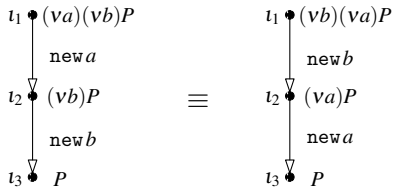
3.2 Structural equivalence on graphs

The set of pi-charts $\{G \mid \iota \bullet P \to^* G\}$ generated by a process P is not preserved by structural congruence of processes, that is, it is not true that if $P \equiv Q$ then P and Q will generate the same charts, or even of the same shape. For example, consider two equivalent processes $(\nu a)(P \mid Q)$ and $P \mid (\nu a)Q$, where $a \notin \mathrm{fn}(P)$. The first process will generate a fresh name a and then branch to P and Q, whereas the second process will branch to P and $(\nu a)Q$, which then can generate the fresh name a. We reconcile these differences by defining a notion of structural congruence on graphs. Let $G \equiv G'$ be the least relation on graphs that is reflexive, symmetric and transitive and that satisfies the following axioms.

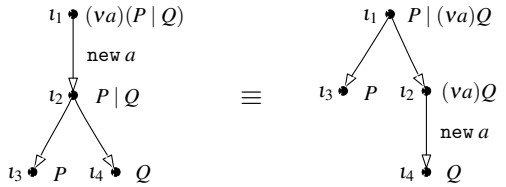
Structural Equivalence on Graphs: $G \equiv H$



SC-COMP-ASSOC



SC-RES



SC-RES-COMP

$$\frac{P \equiv Q}{\iota \bullet P \equiv \iota \bullet Q}$$

SC-CALC

$$\frac{G \equiv G' \quad H \equiv H' \quad L(G,H) \quad L(G',H')}{G \cup H \equiv G' \cup H'} \qquad \text{SC-CONG}$$

We note various properties of equivalent charts in Lemma 3.5. The lemma states that the nodes at the top of equivalent charts are equal up to structural congruence of processes (1), and similarly for the nodes at the bottom (2). Equivalent charts also generate the same fresh names (3), and their corresponding processes are equivalent (4).

Lemma 3.5 *$\forall G$. if G is a pi-chart and $G \equiv H$ then H is a pi-chart and*

(1) $G_\top \equiv H_\top$

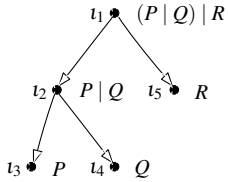
(2) $G_\bot \equiv H_\bot$

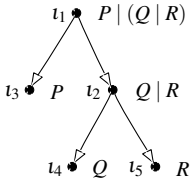
(3) $\mathsf{new}(G) = \mathsf{new}(H)$

(4) $[\![G]\!] \equiv [\![H]\!]$

Proof By induction on the derivation of structural congruence for pi-charts.

SC-COMP-ASSOC

If $G =$ 

and $H =$ 

then:

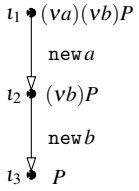
(1) $G_\top = \iota_1 \bullet (P \mid Q) \mid R \equiv \iota_1 \bullet P \mid (Q \mid R) = H_\top$ by STRUCT-PAR-ASSOC

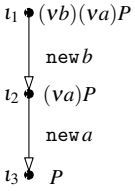
(2) $G_\bot = \iota_1 \bullet P \cup \iota_2 \bullet Q \cup \iota_3 \bullet R = H_\bot$

(3) $\mathsf{new}(G) = \mathsf{new}(H) = \varnothing$

(4) $[\![G]\!] = P \mid Q \mid R = P \mid Q \mid R = [\![H]\!]$

SC-RES

If $G =$ 

and $H =$ 

then:

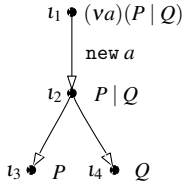
(1) $G_\top = \iota_1 \bullet (\nu a)(\nu b)P \equiv \iota_1 \bullet (\nu b)(\nu a)P = H_\top$ by STRUCT-RES-RES

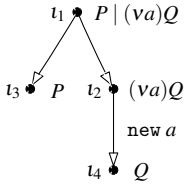
(2) $G_\bot = \iota_3 \bullet P = H_\bot$

(3) $\mathsf{new}(G) = \mathsf{new}(H) = \{a,b\}$

(4) $[\![G]\!] = (\nu a)(\nu b)P = (\nu a)(\nu b)P = [\![H]\!]$

SC-RES-COMP Assume $x \notin \mathrm{fn}(P)$

If $G =$ 

and $H =$ 

then:

(1) $G_\top = \iota_1 \bullet (\nu a)(P \mid Q) \equiv \iota_1 \bullet P \mid (\nu a)Q = H_\top$ by STRUCT-RES-PAR

(2) $G_\bot = \iota_3 \bullet P \cup \iota_4 \bullet Q = H_\bot$

(3) $\mathsf{new}(G) = \mathsf{new}(H) = \{a\}$

(4) $[\![G]\!] = (\nu a)(P \mid Q) = [\![H]\!]$

SC-CALC

If $P \equiv Q$ and $G = \iota \bullet P$ and $H = \iota \bullet Q$ then:

(1) $G_\top = \iota \bullet P \equiv \iota \bullet Q = H_\top$ by (SC-CALC)

(2) $G_\bot = \iota \bullet P \equiv \iota \bullet Q = H_\bot$ by (SC-CALC)

(3) $\mathsf{new}(G) = \mathsf{new}(H) = \varnothing$

(4) $[\![G]\!] = P \equiv Q = [\![H]\!]$

SC-CONG

If $L(G,H)$ and $L(G',H')$ and $G \equiv G'$ and $H \equiv H'$ and $G \cup H$ is a pi-chart then G and H are pi-charts by Lemma 3.6. Therefore ${G_\top} \equiv {G'}_\top$, ${G_\bot} \equiv {G'}_\bot$, $\mathsf{new}(G) = \mathsf{new}(G')$ and $[\![G]\!] \equiv [\![G']\!]$ by induction, and also ${H_\top} \equiv {H'}_\top$, ${H_\bot} \equiv {H'}_\bot$, $\mathsf{new}(H) = \mathsf{new}(H')$ and $[\![H]\!] \equiv [\![H']\!]$ by induction. Let $I = (G_\bot \cap H_\top)$ and $I' = ({G'}_\bot \cap {H'}_\top)$. Therefore $I \equiv I'$ by (SC-CALC). Therefore:

(1) $(G \cup H)_\top = G_\top \cup (H_\top \setminus I) \equiv {G'}_\top \cup ({H'}_\top \setminus I') = (G' \cup H')_\top$ by (SC-CALC)

(2) $(G \cup H)_\bot = H_\bot \cup (G_\bot \setminus I) \equiv {H'}_\bot \cup ({G'}_\bot \setminus I') = (G' \cup H')_\bot$ by (SC-CALC)

(3) $\mathsf{new}(G \cup H) = \mathsf{new}(G) \cup \mathsf{new}(H) = \mathsf{new}(G') \cup \mathsf{new}(H') = \mathsf{new}(G' \cup H') = Z$

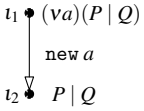
(4) $[\![(G \cup H)_\bot]\!] = [\![H_\bot \cup (G_\bot \setminus I)]\!] = [\![H_\bot]\!] \mid [\![G_\bot \setminus I]\!]$
and $[\![(G' \cup H')_\bot]\!] = [\![{H'}_\bot \cup ({G'}_\bot \setminus I')]\!] = [\![{H'}_\bot]\!] \mid [\![{G'}_\bot \setminus I']\!]$.
Therefore $[\![(G \cup H)_\bot]\!] \equiv [\![(G' \cup H')_\bot]\!]$ by STRUCT-PAR.
Therefore $[\![G \cup H]\!] = (\nu Z)[\![(G \cup H)_\bot]\!] \equiv (\nu Z)[\![(G' \cup H')_\bot]\!] = [\![G' \cup H']\!]$ by STRUCT-RES.

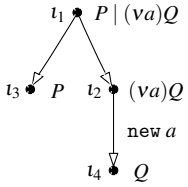
Lemma 3.6 *$\forall G,H$. if $G \cup H$ is a pi-chart and $L(G,H)$ then G and H are pi-charts*

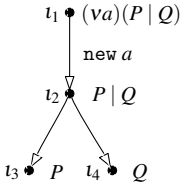
Proof By induction on the derivation of the syntax for pi-charts. For each primitive pi-chart $\mathbf{C}_P$, if $\mathbf{C}_P = G \cup H$ and $L(G,H)$ then either $G = \mathbf{C}_P$ and H is a pi-chart or vice-versa.

We can characterize structural equivalence of processes in terms of the extension relation $G \rightarrow G'$ on graphs and structural equivalence of graphs, as stated in Theorem 3.8. The theorem states that equivalent processes generate equivalent charts, up to housekeeping extensions. Ideally we would like the statement of Theorem 3.8 to hold for $G \equiv H$ rather than the weaker $G \rightarrow_h^* \equiv H$ (recall that $\rightarrow_h$ denotes a "housekeeping" transition involving a

parallel composition or a restriction). Unfortunately, the stronger statement does not hold in general. For a counterexample, consider the two equivalent processes $(\nu a)(P \mid Q)$ and $P \mid (\nu a)Q$, where $a \notin \mathrm{fn}(P)$, and the charts G, H, G' defined as follows.

$G =$ 

$H =$ 

$G' =$ 

We have that $\iota_1 \bullet (\nu a)(P \mid Q)$ can extend to G but $\iota_1 \bullet P \mid (\nu a)Q$ cannot extend to any chart structurally congruent to G. However, $\iota_1 \bullet P \mid (\nu a)Q \longrightarrow\!\!\!\!\!\rightarrow H$ and G can perform an additional housekeeping extension to G' with $G' \equiv H$.

Proposition 3.7 *if $P \equiv Q$ and $\iota \bullet P \rightarrow^* G$ then there is an H such that $\iota \bullet Q \rightarrow^* H$ and $G \rightarrow_h^* \equiv H$*

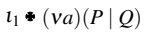
Proof By induction on the derivation of structural congruence for pi-calculus. We assume the following induction hypothesis. If G_0 is a pi-chart and P is a process and $P \equiv Q$ and $P(\iota \bullet P, G_0)$ then:

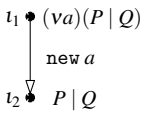
(1) if $G_0 \cup \iota \bullet P \rightarrow^* G$ then $\exists H$ such that $G_0 \cup \iota \bullet Q \rightarrow^* H$ and $G \rightarrow_h^* \equiv H$

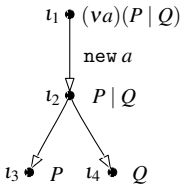
(2) if $G_0 \cup \iota \bullet Q \rightarrow^* H$ then $\exists G$ such that $G_0 \cup \iota \bullet P \rightarrow^* G$ and $H \rightarrow_h^* \equiv G$

The proof exploits the fact that structurally congruent processes generate structurally congruent charts by a series of housekeeping chart extensions. In principle, a housekeeping extension can always happen eventually, regardless of other chart extensions that may occur. We prove this by examining each of the structural congruence rules in turn, in order to ensure consistent use of node identifiers. The base case for part (1) of all the rules is as follows. If $P \equiv Q$ and $\iota_1 \bullet P \cup G_0 \rightarrow^* \iota_1 \bullet P \cup G'$ and $P(\iota_1 \bullet P, G')$ then $\iota_1 \bullet Q \cup G_0 \rightarrow^* \iota_1 \bullet Q \cup G'$ and $P(\iota_1 \bullet Q, G')$. Therefore $\iota_1 \bullet P \cup G' \equiv \iota_1 \bullet Q \cup G'$ by SC-CONG and SC-CALC. The base case for part (2) is similar. Due to lack of space, we show only some representative inductive cases.

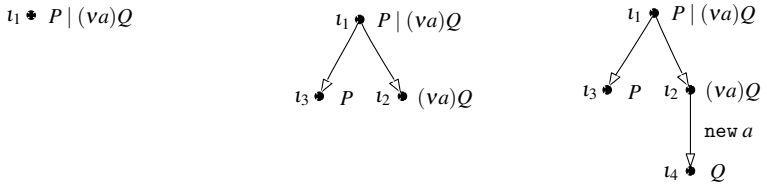
STRUCT-RES-PAR

If $G_1 =$ 

and $G_2 =$ 

and $G_3 =$ 

and $H_1 =$ and $H_2 =$ and $H_3 =$



then part (1) has two remaining cases:

- If $G_1 \cup G_0 \rightarrow^* G_2 \cup G'$ and $L(G_2, G')$ and $\{\iota_2, \iota_3, \iota_4\} \cap \mathsf{I}_{G'} = \varnothing$ then $H_1 \cup G_0 \rightarrow^* H_3 \cup G'$ and $L(H_3, G')$ and $G_2 \cup G' \rightarrow_h G_3 \cup G'$ and $L(G_3, G')$. Therefore $G_3 \cup G' \equiv H_3 \cup G'$ by SC-CONG and SC-RES-COMP.
- Finally, if $G_1 \cup G_0 \rightarrow^* G_3 \cup G'$ and $L(G_3, G')$ then $H_1 \cup G_0 \rightarrow H_3 \cup G'$ and $L(H_3, G')$. Therefore $G_3 \cup G' \equiv H_3 \cup G'$ by SC-CONG and SC-RES-COMP.

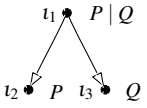
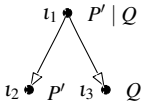
Part (2) is similar.

STRUCT-REFL and STRUCT-TRANS are straightforward

STRUCT-SYMM follows from the induction hypothesis.

STRUCT-PAR

If $P \equiv P'$ and $G_1 = \iota_1 \bullet P \mid Q$ and $H_1 = \iota_1 \bullet P' \mid Q$

and $G_2 =$  and $H_2 =$ 

then part (1) has one remaining case. If $P(G_1 \cup G_0)$ and $\{\iota_2, \iota_3\} \cap G_0 = \varnothing$ then $P(\iota_2 \bullet P, \iota_3 \bullet Q \cup G_0)$. Therefore if $\iota_2 \bullet P \cup \iota_3 \bullet Q \cup G_0 \rightarrow^* G$ we have $\exists H$ such that $\iota_2 \bullet P' \cup \iota_3 \bullet Q \cup G_0 \rightarrow^* H$ and $G \rightarrow_h^* \equiv H$ by induction. Also $L(G_1, G)$ and $L(H_1, H)$. Therefore $G_1 \cup G \equiv H_1 \cup H$ by SC-CONG and SC-CALC.

Part (2) is similar.

Theorem 3.8 *$P \equiv Q$ iff whenever $\iota \bullet P \rightarrow^* G$ there is H with $\iota \bullet Q \rightarrow^* H$ and $G \rightarrow_h^* \equiv H$.*

Proof The forward direction follows by induction on the derivation of structural congruence for the pi-calculus, from Proposition 3.7, while the reverse direction is straightforward. Consider the trivial case where $\iota \bullet P \rightarrow^0 G = \iota \bullet P$. If $\iota \bullet Q \rightarrow^* H$ and $H \equiv G$ we have that $G_\top \equiv H_\top$ by Lemma 3.5. Furthermore, we know that $\iota \bullet Q \rightarrow^* H$ implies $H_\top = \iota \bullet Q$. Therefore $\iota \bullet P \equiv \iota \bullet Q$, so $P \equiv Q$ by Lemma 3.5.

4 Related Work

Starting with Petri [34], there is a substantial literature on graphs as a notation for states of concurrent computations. Examples include process algebras inspired by Petri Nets [3], together with a range of graph-based notations such as [18] and its numerous citations. In the area of process calculi Milner's pi-nets [29] represent pi-calculus processes as graphs, where each node represents a channel and edges to a node represent inputs or outputs on

the channel. Rewrite rules on graphs coalesce nodes after an interaction. Other graph-rewriting based models for the pi calculus include a hypergraph semantics [25] and a term graph semantics [18]. History dependent automata [32] map the entire state space of a pi-calculus process, where each node represents a separate state. The history of names is recorded in the graph, but not the history of computations. Bigraphs [31] are a graphical representation of both the computational and spatial aspects of a process. The graphical stochastic pi-calculus [35] represents a pi-calculus process as a collection of synchronising automata. All these process representations use graphs to represent states of computations, but not the computation history. (However, in certain of the cases one can recover causal relationships [10].) In contrast, a pi-chart represents one of the possible interaction histories of a set of processes, themselves given by syntax trees.

A trace is a sequence of actions performed by a process. In the setting of the pi-calculus, there are several formal definitions of trace, with the aim of defining properties of type systems [39], investigating asynchronous equivalences [6], and defining correspondence assertions [20]. Proved traces [8,38] are decorated with the locations in the term that participated in a transition. Pi-charts enable two-dimensional rendering and record more information, especially regarding restricted names as the subjects and objects of communication.

Various graphical structures are used to define noninterleaving semantics and equivalences of processes; this work has mainly concerned other process calculi and algebras, but recently Varacca and Yoshida [41] develop such a semantics for the pi-calculus using event structures [42]. In contrast, pi-charts are not directly useful (and are not intended) for generating equivalences on processes. The equivalence induced by the set of pi-charts $\{G \mid \iota \bullet P \rightarrow^* G\}$ extending from a process P is syntactic identity, since the process P is embedded in each member of the set. Of course, Theorem 3.4 allows us to reformulate any equivalence relation defined using the interleaving semantics $P \rightarrow^* Q$ in terms of the chart semantics. Our development of structural congruence of graphs, leading to Theorem 3.8, begins the study of equivalences induced by charts.

Cryptographic security protocols are often specified by protocol narrations [1], exemplary sequences of communications of the form "Message n $X \rightarrow Y$: M", meaning that the nth message M of the protocol goes from role X to role Y. A narration itself is essentially an MSC. Some formalisms for security protocols represent protocol runs as MSCs, essentially.

For example, strand spaces [40] are a graphical formalism for protocol narrations, based on strands and bundles. Each strand is a string of inputs and outputs, with implicit name generation, representing a role in the protocol. A bundle is a directed acyclic graph obtained by composing strands, similar to an MSC. Properties of protocols are expressed in terms of occurrences of strands within bundles and "ancestor of" and "earlier than" relations, similar to the causal relations in Section 2.

Crazzolara and Milicia [14] establish explicit formal links between MSCs, formalized as pomsets [37], and the semantics of the Security Protocol Language (SPL) [15]. SPL can be seen as a simple process calculus, with broadcast communication, but without process forking as in the pi-calculus. They define an algorithm for constructing an MSC from any finite trace in the transition semantics of an SPL program. Their main formal result is that the events of such an MSC can be linearized to match the trace and moreover that every linearization of the MSC corresponds to a trace of the original SPL program. Their MSCs

are extracted from an existing semantics for SPL, rather than being defined directly.

5 Conclusion

To summarize, our chart semantics is the first semantics for the pi-calculus based on the idea of message sequence charts. The main benefits of pi-charts compared to a conventional relational semantics are: (1) pi-charts are easier to visualize; and (2) pi-charts can express ancestry and causal dependencies that state-based relational semantics omit.

Although a chart corresponds to a single execution trace, in future we envisage verification tools for proving properties about the set of all charts generated by a given process. For example, this could be useful for validating high-level protocols expressed as pi-calculus processes. In cases where the desired properties do not hold, a visual execution trace representing a counter-example could be presented to the user.

The pi-calculus is used to model programming language features, communication and security protocols and their properties, and more recently, aspects of systems biology (see Appendix A for an example). Hence, the broader significance of our work beyond the pi-calculus is that it forms a formal basis to help visualize and express properties of systems in all of these areas.

Acknowledgements

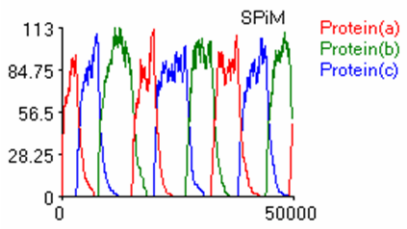
U. Nestmann and G. Winskel advised us on related work. J. Guttman helped us understand the connection between pi-charts and strand spaces.

A A Biological Example

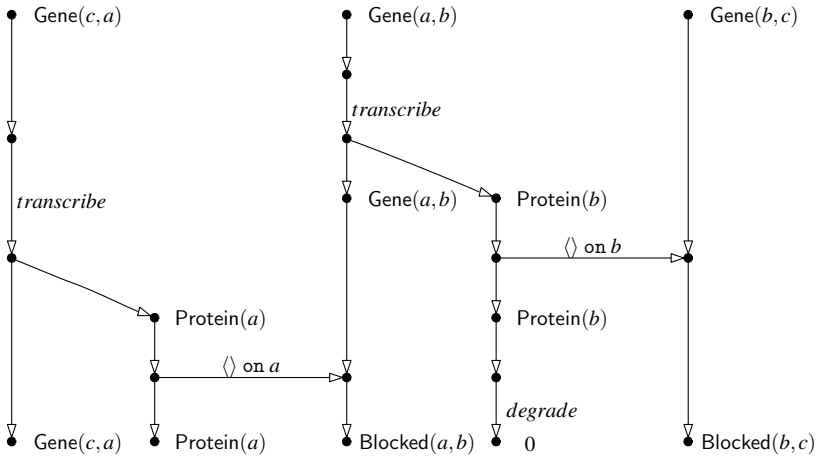
This example shows how pi-charts can be a useful tool for visualising interactions between stochastic pi-calculus models of biological systems. We use the same pi-calculus syntax and reduction rules from Sections 2 and 3, enriched with a stochastic extension along the lines of [35]. Stochastic behaviour is incorporated into the calculus by associating each channel a with a corresponding interaction rate given by $\rho(a)$, and associating each action τ_r with a delay rate r. The rates are used as the basis for a stochastic simulation algorithm, which calculates the probability of all possible reductions at each step and stochastically chooses the next reduction based on these probabilities.

Consider the following network of three genes that mutually repress each other, with definitions for $\mathsf{Gene}(a,b)$, $\mathsf{Blocked}(a,b)$ and $\mathsf{Protein}(b)$ based on [35]:

$$\begin{aligned}
&\mathsf{Gene}(a,b) := \tau_{transcribe}.(\mathsf{Gene}(a,b) \mid \mathsf{Protein}(b)) \\
&\qquad\qquad + a().\mathsf{Blocked}(a,b) \\
&\mathsf{Blocked}(a,b) := \tau_{unblock}.\mathsf{Gene}(a,b) \\
&\mathsf{Protein}(b) := \overline{b}\langle\rangle.\mathsf{Protein}(b) \\
&\qquad\qquad + \tau_{degrade} \\
\\
&\mathsf{Gene}(a,b) \mid \mathsf{Gene}(b,c) \mid \mathsf{Gene}(c,a)
\end{aligned}$$



The $\mathsf{Gene}(a,b)$ is parameterised by its promoter region a, together with the promoter region b that is recognised by its transcribed proteins. The gene can perform one of two actions. Either it can transcribe a $\mathsf{Protein}(b)$ by doing a stochastic delay at rate *transcribe*, after which the new protein is executed in parallel with the gene, or it can block by doing an input on its promoter region a. The blocked gene can then unblock by doing a stochastic delay at rate *unblock*. The $\mathsf{Protein}(b)$ can repeatedly do an output on the promoter region b, or it can decay at rate *degrade*. According to the reduction rules of the calculus, the output $\overline{b}\langle\rangle$ of the transcribed protein can interact with the input $b()$ of a $\mathsf{Gene}(b,c)$, which becomes blocked as a result. The three genes $\mathsf{Gene}(a,b)$, $\mathsf{Gene}(b,c)$ and $\mathsf{Gene}(c,a)$ can mutually repress each other, since $\mathsf{Gene}(a,b)$ produces a protein that can block $\mathsf{Gene}(b,c)$, which produces a protein that can block $\mathsf{Gene}(c,a)$, which produces a protein that can block $\mathsf{Gene}(a,b)$, completing the cycle. This mutual repression gives rise to alternate oscillation of protein levels, as shown in the above simulation plot, in which the vertical axis represents the number of processes and the horizontal axis represents the simulation time. The results were obtained with equal rates for channels a,b,c such that $\rho(a) \gg \mathit{transcribe} \gg \mathit{degrade} > \mathit{unblock}$. However, the plots themselves give no indication as to what actually causes the oscillations to occur. Such a question is fundamental to understanding the behaviour the system, and pi-charts can help to provide a partial answer. An execution trace for the system is represented by the following pi-chart, which shows how the system can evolve starting from one of each gene. The visual representation of causality in the pi-chart helps to clarify the sequence of execution steps leading to the first oscillation cycle.



The chart shows how one of the genes, in this case $\mathsf{Gene}(a,b)$, transcribes a $\mathsf{Protein}(b)$, which immediately blocks $\mathsf{Gene}(b,c)$. $\mathsf{Gene}(c,a)$ transcribes $\mathsf{Protein}(a)$ soon after, which blocks $\mathsf{Gene}(a,b)$.The $\mathsf{Gene}(a,b)$ and $\mathsf{Gene}(b,c)$ both remain blocked, waiting for a slow *unblock* delay to fire, while $\mathsf{Gene}(c,a)$ is able to freely produce $\mathsf{Protein}(a)$ and start the first oscillation cycle.

We have implemented a prototype stochastic simulator that automatically generates a pi-chart during a given simulation run. The prototype was implemented as a simple extension to the SPiM simulator,[5] by exporting the execution history of a simulation to a file using the DOT syntax [19]. The DOT layout engine is then used to automatically render the file as a pi-chart. The generated charts can be quite large, but it is relatively easily to scroll and zoom through the charts to a time point of particular interest in the simulation. For the above biological example one can focus on the sequence of transitions leading up to a switch in oscillation cycles, which can be quite informative.

In general, pi-charts seem to be a convenient way of visualising and debugging the behaviour of concurrent biological systems, and initial reactions from biologists have so far been positive. We plan to include a pi-chart debugging option in the next release of the SPiM simulator, so that biologists can experiment with generating their own charts from a range of models.

B Expressing the Bounds Guaranteed by a Type System

We present a synthesis of some existing type systems, including groups (or sorts) [30], group creation [12], and usage bounds [24]. A channel type T takes the general form $g\ ?i\ !o\ [T_1,\ldots,T_n]$. We say g is the *group* of the type, and of names belonging to the type. Groups indicate different usages, for example, REQ or RES. A name x of type T is a channel conveying tuples of names with types T_1, ..., T_n. The *multiplicities* i and o are upper bounds on the number of uses of x for input and output.

Group creation $(\nu grp\ g)P$ makes a fresh group g for use within P. Groups are represented as names, but well-typed processes cannot send them on channels. Hence, group

[5] SPiM is available at http://research.microsoft.com/~aphillip/spim/.

creation helps structure processes by confining the flow of names belonging to a group. In particular, if a process $O \mid (\nu grp\ g)P$ is well-typed and there is a name x in group g, then the name x is communicated only between descendants of P—the lexical scope of g—and cannot flow to descendants of O.

Our point here is not the type system itself, an assembly of variations of existing components, but rather to show that pi-charts can conveniently express both the usage bounds induced by multiplicities and the secrecy properties induced by group creation. The original statement of the latter [12, Proposition 3] relies on an informal notion of process derivation; our statement in terms of the "ancestor of" relation is completely formal.

We proceed with a terse presentation of the type system. Further explanations and examples are in the original publications [12,24,30].

Groups and Types:

g, h	group: subset of the set of the names
$\mu, i, o ::= 0 \mid 1 \mid \omega$	multiplicity
$T ::= g\ ?i\ !o\ [T_1, \ldots, T_n]$	polyadic channel type ($n \geq 0$)
$m ::= g \mid (x : T)$	item: either a group, or a name with a type
$E ::= \varnothing, m_1, \ldots, m_n$	typing environment: finite list of items

$\mathrm{dom}(\varnothing) := \varnothing \quad \mathrm{dom}(E, g) := \mathrm{dom}(E) \cup \{g\} \quad \mathrm{dom}(E, x : T) = \mathrm{dom}(E) \cup \{x\}$

Our pi-calculus syntax is untyped, but we place type and group annotations on τ prefixes, both to guide typechecking, and to record typing information in the pi-chart semantics. We take the algebra of annotations $\mathbf{A}$ to be the set of items, so that we can write $\tau_g.P$ and $\tau_{x:T}.P$. Let *typed name restriction* be $(\nu x : T)P := (\nu x)\tau_{x:T}.P$ and *group creation* be $(\nu grp\ g)P := (\nu g)\tau_g.P$. Every chart extending from $(\nu x : T)P$ and reaching P includes edges $\iota \xrightarrow{\mathtt{new}\ x} \iota'$ and $\iota' \xrightarrow{x:T} \iota''$ and node $\iota'' \bullet P$. Similarly, every chart extending from $(\nu grp\ g)P$ and reaching P includes $\iota \xrightarrow{\mathtt{new}\ g} \iota'$, $\iota' \xrightarrow{grp\ g} \iota''$, and $\iota'' \bullet P$.

Let the addition $\mu + \mu'$ of two multiplicities be the commutative function satisfying the equations $\mu + 0 = \mu$ and $\mu + \omega = \omega$ and $0 + 1 = 1$ and $1 + 1 = \omega$. The addition functions on types, items, and environments are the least partial functions to satisfy the following equations. They are all associative and commutative.

Type, Item, and Environment Addition: $T + T' \quad m + m' \quad E + E'$

$(g\ ?i\ !o\ [T_1, \ldots, T_n]) + (g\ ?i'\ !o'\ [T_1, \ldots, T_n]) := g\ ?(i + i')\ !(o + o')\ [T_1, \ldots, T_n]$
$g + g := g$
$(x : T) + (x : T') := x : (T + T')$
$(\varnothing, m_1, \ldots, m_n) + (\varnothing, m'_1, \ldots, m'_n) := (\varnothing, m_1 + m'_1, \ldots, m_n + m'_n)$

We assume a relation between process constants and lists of groups and types describing their parameters. Specifically, for each definition $\mathsf{A}(x_1, \ldots, x_n) := P$, we assume that the constant A is related to a list of group parameters $h_1, \ldots, h_m$ and a list of types $T_1, \ldots, T_n$. We write this as $\mathsf{A}[h_1, \ldots, h_m, x_1 : T_1, \ldots, x_n : T_n]$.

The following rules define four judgments: $E \vdash \diamond$ means that the environment E is well-formed; $E \vdash m$ means that the item m occurs in E; $E \vdash T$ means that the type T is

well-formed in E; and $E \vdash P$ means that the process P is well-formed in E.

Typing Rules: $E \vdash \diamond \quad E \vdash m \quad E \vdash T \quad E \vdash P$

ENV-$\varnothing$
$$\frac{}{\varnothing \vdash \diamond}$$

ENV-GROUP
$$\frac{E \vdash \diamond \quad g \notin \mathrm{dom}(E)}{E, g \vdash \diamond}$$

ENV-NAME
$$\frac{E \vdash T \quad x \notin \mathrm{dom}(E)}{E, x : T \vdash \diamond}$$

LOOKUP
$$\frac{\varnothing, m_1, \ldots, m_n \vdash \diamond \quad i \in 1..n}{\varnothing, m_1, \ldots, m_n \vdash m_i}$$

TYPE
$$\frac{E \vdash g \quad E \vdash T_1 \quad \ldots \quad E \vdash T_n}{E \vdash g\ ?i\ !o\ [T_1, \ldots, T_n]}$$

PROC-ZERO
$$\frac{E \vdash \diamond}{E \vdash \mathbf{0}}$$

PROC-IN
$$\frac{E_0 \vdash x : g\ ?1\ !0\ [T_1, \ldots, T_n] \quad E_1, y_1 : T_1, \ldots, y_n : T_n \vdash P \quad E = E_0 + E_1 \text{ defined}}{E \vdash x(y_1, \ldots, y_n).P}$$

PROC-OUT
$$\frac{E_0 \vdash x : g\ ?0\ !1\ [T_1, \ldots, T_n] \quad E_i \vdash y_i : T_i \quad \forall i \in 1..n \quad E_{n+1} \vdash P \quad E = E_0 + \cdots + E_{n+1} \text{ defined}}{E \vdash \overline{x}\langle y_1, \ldots, y_n \rangle.P}$$

PROC-NOTE
$$\frac{E \vdash m \quad E \vdash P}{E \vdash \tau_m.P}$$

PROC-CHOICE
$$\frac{E \vdash M \quad E \vdash N}{E \vdash M + N}$$

PROC-RES-GROUP
$$\frac{E, g \vdash P}{E \vdash (\nu grp\ g)P}$$

PROC-RES
$$\frac{E, x : T \vdash P}{E \vdash (\nu x : T)P}$$

PROC-PAR
$$\frac{E_1 \vdash P_1 \quad E_2 \vdash P_2 \quad E = E_1 + E_2 \text{ defined}}{E \vdash P_1 \mid P_2}$$

PROC-CONST
$$\frac{\mathsf{A}[h_1, \ldots, h_m, x_1 : T_1, \ldots, x_n : T_n] \quad \sigma = \{{}^{g_j}/_{h_j} \mid j \in 1..m\} \quad E \vdash \diamond \quad E \vdash g_j \quad \forall j \in 1..m \quad E_i \vdash c_i : T_i\sigma \quad \forall i \in 1..n \quad E = E_1 + \cdots + E_n \text{ defined}}{E \vdash \mathsf{A}(c_1, \ldots, c_n)}$$

We assume that $h_1, \ldots, h_m, x_1 : T_1, \ldots, x_n : T_n \vdash P$ for each definition $\mathsf{A}(x_1, \ldots, x_n) := P$ where $\mathsf{A}[h_1, \ldots, h_m, x_1 : T_1, \ldots, x_n : T_n]$.

Theorem B.1 *Suppose $E \vdash [\![G_\top]\!]$, G is a pi-chart, and $T = g\ ?i\ !o\ [T_1, \ldots, T_n]$.*

(1) *If $\iota_1 \xrightarrow{x:T} \iota_2$ then the number of communications on x in G is no more than $\min(i, o)$.*

(2) *If $\iota_1 \xrightarrow{g} \iota_2$ and $\iota_3 \xrightarrow{x:T} \iota_4$ then ι_2 [Ancestor] ι_3.*
Moreover, if $\iota_5 \xrightarrow{\langle \widetilde{y} \rangle \text{ on } z} \iota_6$ and $x \in \mathrm{fn}(\widetilde{y}, z)$ then ι_2 [Ancestor] ι_5 and ι_2 [Ancestor] ι_6.

We can explain the secrecy property of group creation by appeal to this theorem. Suppose that $E \vdash O \mid (\nu grp\ g)P$, and consider any pi-chart G such that $G_\top = \iota \bullet (O \mid (\nu grp\ g)P)$

for some ι. Such a G represents an arbitrary interaction between the process O and the process $(\nu grp\ g)P$. Unless G is a singleton, in which case it includes no interactions, it must include an instance of the primitive chart for parallel composition, with edges $\iota \rightarrowtriangle \iota'$ and $\iota \rightarrowtriangle \iota''$, and nodes $\iota' \bullet O$ and $\iota'' \bullet (\nu grp\ g)P$. As discussed above, if P is reached, there must be edges $\iota'' \xrightarrow{\texttt{new}\ g} \iota_1$, $\iota_1 \xrightarrow{grp\ g} \iota_2$, and a node $\iota_2 \bullet P$. By Lemma 2.5, no descendant of $\iota' \bullet O$ is a descendant of $\iota_2 \bullet P$, and the converse. If a name x of group g is created, there must be an edge $\iota_3 \xrightarrow{x:T} \iota_4$, where g is the group of T. By Theorem B.1(ii), ι_2 [Ancestor] ι_3, that is, a descendant of P creates the name x. Now, consider any communication of x, that is, consider any edge $\iota_5 \xrightarrow{\langle \widetilde{y} \rangle\ \texttt{on}\ z} \iota_6$ with $x \in \mathrm{fn}(\widetilde{y})$. By Theorem B.1(ii), ι_2 [Ancestor] ι_5 and ι_2 [Ancestor] ι_6, that is, both the sender ι_5 and the receiver ι_6 of the tuple $\widetilde{y}$ containing x are descendants of P. Additionally, the theorem implies that all communications on the channel x itself are between descendants of P.

Hence, pi-charts directly formalize the intention that "channels of group g are forever secret outside the initial scope of $(\nu grp\ g)$" [12].

References

[1] M. Abadi. Security protocols and their properties. In *Foundations of Secure Computation*, NATO Science Series, pages 39–60. IOS Press, 2000.

[2] R. Alur, G. J. Holzmann, and D. Peled. An analyzer for message sequence charts. *Software Concepts and Tools*, 17(2):70–77, 1996.

[3] E. Best, R. R. Devillers, and M. Koutny. The box algebra = petri nets + process expressions. *Inf. Comput.*, 178(1):44–100, 2002.

[4] K. Bhargavan, C. Fournet, A. D. Gordon, and R. Pucella. TulaFale: A security tool for web services. In *FMCO'03*, volume 3188 of *LNCS*, pages 197–222. Springer, 2004.

[5] E. Bonelli, A. Compagnoni, and E. Gunter. Correspondence assertions for process synchronization in concurrent communications. *JFP*, 15(2):219–147, 2004.

[6] M. Boreale, R. De Nicola, and R. Pugliese. Trace and testing equivalence on asynchronous processes. *Information and Computation*, 172(2):139–164, 2002.

[7] M. Boreale and D. Sangiorgi. A fully abstract semantics of causality in the π-calculus. *Acta Informatica*, 35:353–400, 1998.

[8] G. Boudol and I. Castellani. Concurrency and atomicity. *TCS*, 59:25–84, 1988.

[9] G. Boudol and I Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fundamenta Informaticae*, 4(XI):433–452, 1988.

[10] R. Bruni, H. C. Melgratti, and U. Montanari. Event structure semantics for nominal calculi. In *CONCUR*, pages 295–309, 2006.

[11] M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *Theoretical Aspects of Computer Software (TACS 2001)*, volume 2215 of *LNCS*, pages 38–63. Springer, 2001.

[12] L. Cardelli, G. Ghelli, and A. D. Gordon. Secrecy and group creation. *Information and Computation*, 196(2):127–155, 2005.

[13] I. Castellani. Process algebras with localities. In *Handbook of Process Algebra*, chapter 15, pages 945–1045. Elsevier, 2001.

[14] F. Crazzolara and G. Milicia. Graphical descriptions of security protocols. In *COnstraint & LOgic Programming in Security (COLOPS 2003)*, 2003.

[15] F. Crazzolara and G. Winskel. Events in security protocols. In *Eight ACM Conference on Computer and Communications Security (CCS'2001)*, 2001.

[16] P. Degano and C. Priami. Non interleaving semantics for mobile processes. *Theoretical Computer Science*, 216:237–270, 1999.

[17] P. Degano and C. Priami. Enhanced operational semantics. *ACM Computing Surveys*, 2(33):135–176, 2001.

[18] F. Gadducci. Term graph rewriting for the pi-calculus. In Atsushi Ohori, editor, *APLAS*, volume 2895 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2003.

[19] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software—Practice and Experience*, 30(11):1203–1233, 2000.

[20] A. D. Gordon and A. Jeffrey. Typing correspondence assertions for communication protocols. *Theoretical Computer Science*, 300:379–409, 2003.

[21] ITU. *Message Sequence Chart (MSC)*, 1999. Recommendation Z.120.

[22] ITU. *Specification and Design Language (SDL)*, 1999. Recommendation Z.100.

[23] A. S. A. Jeffrey and J. Rathke. A fully abstract may testing semantics for concurrent objects. *Theoretical Computer Science*, 338:17–63, 2005.

[24] N. Kobayashi, B. Pierce, and D. Turner. Linearity and the pi-calculus. *TOPLAS*, 21(5):914–947, 1999.

[25] B. König. A graph rewriting semantics for the polyadic pi-calculus. In *Proc. of GT-VMT '00 (Workshop on Graph Transformation and Visual Modeling Techniques)*, pages 451–458. Carleton Scientific, 2000.

[26] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[27] S. Mauw and M. A. Reniers. An algebraic semantics of basic message sequence charts. *The Computer Journal*, 37(4), 1994.

[28] S. Mauw, M.A. Reniers, and T.A.C. Willemse. Message Sequence Charts in the software engineering process. In S.K. Chang, editor, *Handbook of Software Engineering and Knowledge Engineering*. World Scientific, 2000.

[29] R. Milner. Pi-nets: A graphical form of π-calculus. In *ESOP'94*, pages 26–42, 1994.

[30] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. CUP, 1999.

[31] R. Milner. Bigraphical reactive systems: Basic theory. Technical Report 523, University of Cambridge Computer Laboratory, 2001.

[32] U. Montanari and M. Pistore. History-dependent automata: An introduction. *Lecture Notes in Computer Science*, 3465:1–28, 2005.

[33] Object Management Group. *Unified Modeling Language*. At `http://www.uml.org`.

[34] C. Petri. Fundamentals of a theory of asynchronous information flow. In *IFIP Congress '62*, pages 386–390. North Holland, 1962.

[35] A. Phillips, L. Cardelli, and G. Castagna. A graphical representation for biological processes in the stochastic pi-calculus. *Transactions in Computational Systems Biology*, 4230:123–152, 2006.

[36] A. Phillips, N. Yoshida, and S. Eisenbach. A distributed abstract machine for boxed ambient calculi. In *European Symposium on Programming*, LNCS. Springer, April 2004.

[37] V. Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[38] C. Priami. *Enhanced Operational Semantics for Concurrency*. PhD thesis, Pisa, 1996.

[39] D. Sangiorgi and D. Walker. *The π-calculus: A theory of mobile processes*. CUP, 2001.

[40] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.

[41] D. Varacca and N. Yoshida. Typed event structures and the pi-calculus. In *Mathematical Foundations of Programming Semantics*, ENTCS. Elsevier, 2006.

[42] G. Winskel. *Events in Computation*. PhD thesis, University of Edinburgh, 1980.