View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

The Journal of Logic and Algebraic Programming 77 (2008) 92–130

Contents lists available at ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

Pathway analysis for BioAmbients

Henrik Pilegaard*, Flemming Nielson, Hanne Riis Nielson

DTU Informatics, Technical University of Denmark, Denmark

ARTICLE INFO

Available online 18 July 2008

Keywords:
Static analysis
Pathway analysis
Flow logic
Monotone frameworks
Control flow analysis
Data flow analysis
Process calculus
BioAmbients
Systems Biology
LDL degradation pathway

ABSTRACT

Systems Biology aims for a holistic understanding of biological processes. In order to make this understanding operational and testable it can be recorded into formal process calculus models. This is a difficult task, however, because such formal models and their, often infinitely many, consequences are hard to enumerate and understand. In this paper we define a *pathway analysis*, based on static analysis techniques from programming languages, and show how it can be used to establish useful, finite, approximations to the set of causal consequences of models. The Pathway Analysis can be used to great advantage in all phases of the modelling approach – serving as the basis of debugging during model development, postdiction during model validation, and, finally, prediction during model guided drug design.

© 2008 Elsevier Inc. All rights reserved.

1. Introduction

The field of *Systems Biology* [16,15] seeks a holistic understanding of biological systems. By studying the relationships and interactions between biological entities or subsystems, such as gene and protein signalling networks, metabolic pathways, or cells, it is hoped that a precise model of the whole system can be developed. This work is hypothesis driven – *computational models*, firmly based on knowledge from molecular biology, are proposed and then iteratively refined in order to explain the systems that lead to observable phenotypes.

The goal of this development is the scenario depicted in Fig. 1, where *in vivo*/*in vitro* experiments, such as those involved in drug design, can potentially be carried out *in silico*. Arguably, such a *drylab* would have an enormous impact but actually obtaining one is also quite complex; as illustrated by Fig. 2 it is convenient to talk about three distinct phases of development:

The modelling phase deals with the formalisation of hypotheses.

The validation phase deals with the postdiction of experimental results.

The analysis phase deals with the prediction of experimental results.

Throughout development quality must be ensured. In the long run the quality of a model is determined by its ability to predict the physical reality that it describes. In the short run, however, and in particular during model development, the quality is more appropriately determined by the absence of contradictions to existing knowledge. In both cases we can only

* Corresponding author. Tel.: +45 45253346.
E-mail address: hepi@imm.dtu.dk (H. Pilegaard).

1567-8326/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2008.05.006

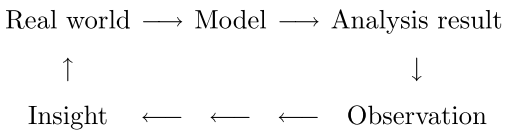


Fig. 1. The goal of model based observation.

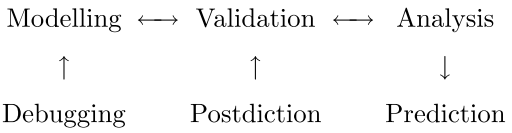


Fig. 2. Phases in model development.

measure the quality if able to determine the consequences of a given model: What behaviour **is** and what behaviour is **not** actually described by the model?

In this paper we shall address this question by defining a *pathway analysis* that, given a formal model, P, computes a finite over-approximation to the full set of (possibly infinite) reaction sequences that P describes. The computed approximations are *safe*: Whenever an interesting state might be reachable the analysis will show it and reveal the sequences of events that might lead there. If the model is faithful, one or more of these sequences might pinpoint an actual biological pathway. Most importantly, however, states and reaction sequences that do not appear in the analysis result are *not* realisable. Thus the analysis is immensely useful in demonstrating, e.g., that undesirable reactions, perhaps part of a dysfunctional pathway, cannot arise.

Background. Advances in Life Science have revealed that the behaviour of a living cell emerges as a result of complicated interactions among a set of biochemical agents. The resulting modern view, advanced by the post-genomic science of Systems Biology, is that life *is* the complex behaviour that emerges from systems within systems of interacting reactive entities.

In the Computer Science area of Concurrency Theory the notion of computation is regarded in much the same way – it *is* the complex behaviour that emerges from systems within systems of reactive concurrent processes. Hence Computer Scientists have spent three decades inventing modelling formalisms and analysis techniques in order to capture and understand the behaviour of such systems.

The similarity of the two domains implies a significant potential for cross-fertilisation. Among the first to take advantage of this were Regev and Shaphiro [40,39,38], who pioneered the use of Process Calculi for the modelling of Biological Systems. Works by Priami [37,36], Danos [8,12], Cardelli [39,5,6], Hillston [4], Harel [14], ourselves [32,35,34], and others have extended this line of work.

Here we extend this line of work by basing our approach on the BioAmbients calculus, developed by Regev et al. in [39]. It is a biologically oriented version of the Ambient Calculus [7] – a well known process calculus used to describe concurrent, distributed, and mobile systems. The variant retains a notion of spatial ambient boundaries from the original calculus, which allows for models that preserve the spatial structure of the bio-domain. For this reason it is well suited for the description of cellular transport networks. BioAmbients also incorporates a notion of channeled communication in the manner of the π-calculus [21] – a language that is widely used for the modelling of cellular signalling and regulation networks [3,40,17]. It is woth mentioning that BioAmbients is actually a proper superset of the π-calculus and hence the proposed pathway analysis also applies to this language.

Indeed, the analysis techniques that we present in this paper also have their roots in Computer Science, specifically in the area of optimising compilers: In order to make statements about potentially undecidable properties of models we take the approach of *static program analysis* and compute *safe approximations* rather than precise results [23]. As shown in Fig. 3 we consider an approximation safe when it is *provably* either an under- or over-approximation as this allows us to make safe statements about the presence or absence of properties, respectively.

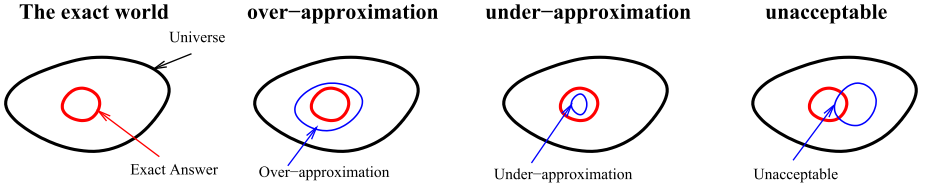


Fig. 3. The nature of static approximations.

Technically, we obtain our pathway analysis by combining the approach of *static control flow analysis* for process calculi with a recent line of work by Nielson and Nielson [30,31], which concerns the use of *static data flow analysis* in conjunction with process calculi.

Static *control flow analyses* (CFAs) have previously been used successfully for computing safe approximations to the sets of spatial configurations reachable by BioAmbients models [32,35,25]. Given a program $P_\star$ these analyses efficiently compute containment graphs that are guaranteed to contain all possible run-time configurations as sub-graphs.

Mainly concerned with properties of the *configurations* the CFAs tell very little about the properties of the *transitions*. However, knowing *what* configurations may be reachable but not *how* is clearly inadequate when trying to meet the outlined goals. This is the issue we shall address by developing an analysis that focuses on transitions rather than configurations.

Overview. In Section 2 we describe the BioAmbients formalism in more detail. Then, in Section 3 we specify a CFA in the tradition of [32,35] intended to support our pathway analysis by providing *safe approximations* to the sets of prefix pairs that may react at run-time. In Section 4 we adapt notions from *monotone frameworks* [23,31] in order to safely approximate how the *extended multiset* of *exposed reactive prefixes* of a model may evolve as prefix pairs react. After this, in Section 5 we define a *worklist algorithm* that computes *finite automata* approximations to model behaviour. Finally, in Section 6 we treat a larger case study regarding, *the LDL degradation pathway*, the prototypical example of large molecule internalisation and transport.

Throughout the technical developments we shall illustrate our developments by applying them to a BioAmbients model P_{eat}, which concerns the absorption of nutrients from larger 'food' compounds.

2. BioAmbients

The BioAmbients calculus of Regev et al. [39,38,5] is a variant of Mobile Ambients [7] designed to model biological systems. The calculus is rich in terms of primitives and includes constructs for many aspects of the biological domain.

Firstly, the calculus preserves the notion of *ambients* as bounded mobile sites of activity that may nest hierarchically. This provides an intuitive means for modelling both the chemically active *membrane-bound compartments* that are ubiquitous in eukaryotic cells and the *molecular compartments* that arise as a result of complexation. In order to make these spatial abstractions operational the calculus incorporates a set of *capability primitives* for the modelling of processes that alter the local nesting hierarchy. In the context of an ambient-as-molecular-compartment abstraction, e.g., capabilities may be used to model the formation and breaking of complexes. In the context of an ambient-as-membrane-bound-compartment abstraction, they may be used to model such phenomena as phago-/endo-cytosis, exo-cytosis, and membrane fusion.

Secondly, the calculus incorporates the notion of *channelled communication* from the π-calculus. This allows simpler biological entities (i.e. proteins, RNA, and DNA) and their *interaction networks* to be modelled as networks of interacting π-style processes [40]. In order to make this network abstraction operational in the context of spatial abstraction the calculus incorporates a set of *action primitives* for the modelling of processes that interact across ambient boundaries as well as locally. In the context of an ambient-as-molecular-compartment abstraction, for example, such primitives may be used to model complexes that act as enzymes or complexes that undergo internal formation changes. In the context of an ambient-as-membrane-bound-compartment abstraction, on the other hand, they may be used to model cross-membrane interactions or interactions confined to the local environment of an organelle.

The set of control structures for processes is slightly larger than what is traditionally studied for Mobile Ambients. It includes non-deterministic (external) choice as well as a general recursion construct in the manner of CCS [20]. This helps to achieve an intuitive modelling of biological phenomena, while retaining a great deal of precision.

What brings all of these elements together and completes the biological abstraction is a reaction semantics in the style of the Chemical Abstract Machine [2]. The interpretation thus provided is exactly the right one: A BioAmbients model describes a chemical soup of reactive entities distributed over some spatial configuration. Two such entities may react (synchronously) if they are close to one another and exhibit *reactive domains* (modelled by capability or action prefixes) that are complementary in terms of *shape* (channel name) and *purpose* (matching action/co-action or capability/co-capability).

Due to the multiple modelling roles of ambients the resulting formalism is characterised by generality rather than specialisation. The modelling of complexes in BioAmbients, e.g., does not describe how the active domains of proteins and complexes are hidden and exposed as a result of interactions; the κ-calculus [10,11] was introduced to cover this aspect. Similarly, the modelling of membranes in terms of ambients neglects that membranes are oriented and that interactions always preserves this property; this provoked the invention of Brane calculus [6,12]. Finally, the calculus assumes that biological interactions only take place when active domains exhibit perfect complementarity; the β-binders calculus [36] was first introduced to overcome this so-called key-lock assumption.

2.1. Syntax

The full syntax of BioAmbients is defined in Fig. 4, where we write $P \in \mathbf{Proc}$ for *processes* and $M \in \mathbf{CA}$ for *capabilities* and *actions*. Capabilities and actions are based on *names* as we know them from CCS and the π-calculus [21]. We shall make a distinction between names introduced by the restriction operator $(n)\,P$, which we consider to be *constants* ($n,m \in \mathbf{C}$), and names introduced by communication capabilities, e.g. the p in $n?\{p\}$, which we consider to be *variables* ($p,q \in \mathbf{V}$). We require the sets of constants and variables to be mutually disjoint so that we are dealing with a countable set $\mathbf{Name} = \mathbf{C} \uplus \mathbf{V}$ of names

$$
\begin{array}{rrl}
P \in \mathbf{Proc} ::= & (n)\,P & \text{Name restriction} \\
| & \lceil P \rceil^{\mu} & \text{Ambient boundary} \\
| & P \mid P' & \text{Parallel composition} \\
| & \sum_{i \in I} M_i^{\ell_i} . P_i & \text{Summation (guarded choice)} \\
| & \mathsf{rec}\, X.\, P & \text{Recursive process (defining } X \triangleq \mathsf{rec}\, X.\, P) \\
| & X & \text{Process identifier}
\end{array}
$$

$$
\begin{array}{rrl}
M \in \mathbf{CA} ::= & & \textbf{Capability primitives} \\
| & \mathsf{enter}\ x \mid \mathsf{accept}\ x & \text{enter movement} \\
| & \mathsf{exit}\ x \mid \mathsf{expel}\ x & \text{exit movement} \\
| & \mathsf{merge}{-}\ x \mid \mathsf{merge}{+}\ x & \text{merge movement} \\
 & & \textbf{Action primitives} \\
 & x!\{y\} \mid x?\{p\} & \text{local communication} \\
| & x\ !\{y\} \mid x\hat{}?\{p\} & \text{parent to child communication} \\
| & x\hat{}!\{y\} \mid x\ ?\{p\} & \text{child to parent communication} \\
| & x\#!\{y\} \mid x\#?\{p\} & \text{sibling to sibling communication}
\end{array}
$$

Fig. 4. Syntax of BioAmbients.

and we shall write x,y for elements of this set. Finally, we shall assume a countable set of *process identifiers* (X,$Y \in$ **Pid**). Note that we use the heavy brackets $\lceil$ and $\rceil$ to represent ambient boundaries; the ordinary brackets, [and], are reserved for the notion of *substitution* defined in Section 2.2.

Processes are analogous to biological systems of interacting reactive entities. We shall now explain the calculus primitives and elaborate on this analogy:

Inaction, $\mathbf{0}$ (arises as the nullary special case of summation) denotes a process that can do nothing. Technically, this is the terminal process that is the end of all things (including recursive analysis). Biologically, this is a system that is completely depleted of reactive entities; we shall think of it merely as superfluous solvent, e.g., water, that may dissipate by evaporation.

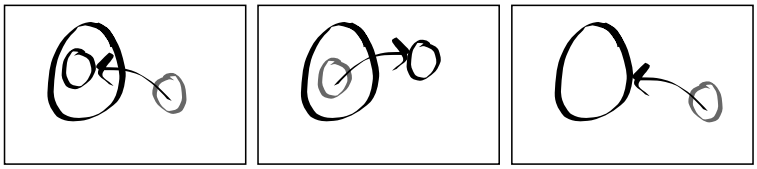
The *prefix*, $M^{\ell} . P$ (arises as the unary special case of summation) denotes a process that is capable of participating in the reaction identified by the primitive M; exercising M turns the process into the continuation P. This corresponds to a biological entity, such as a protein, that exposes a single binding site and is altered in some way by the corresponding reaction. Note that we draw upon a countable set of *labels*, $\ell \in$ **Lab**, in order to uniquely annotate each prefix. As will later become clear these labels have no semantic significance – we shall merely use them as convenient pointers into processes when defining our analyses in Sections 3 and 4.

The *summation*, $\sum_{i \in I} M_i^{\ell_i} . P_i$, generalises the prefix and, thus, denotes a process that is capable of participating in any one of the $k = |I|$ reactions identified by the primitives M_i. Again, exercising some M_j turns the process into the corresponding continuation P_j; the remaining primitives and the corresponding continuations disappear. This corresponds to a biological entity, such as a protein, that exposes k distinct binding sites and is altered in some way when one of them engages in reaction. Note that Regev et al. [39], who are concerned with the computation of rates for a stochastic semantics, disallows the mixing of capabilities and actions within summations. Here we have no such concerns and do not uphold this separation.

The *recursive process*, $\mathsf{rec}\, X.\, P$, models recurrent behaviour. Technically, the construct denotes a process that behaves as $P[^{\mathsf{rec}\, X.\, P}/_{X}]$, i.e. as P with every occurrence of X replaced by $\mathsf{rec}\, X.\, P$. Biologically, this corresponds to entities that have cyclic behaviour. One example is an entity $\mathsf{rec}\, X.\, M^{\ell} . X$, such as an enzyme, that may participate in the same reaction – identified by M – over and over. Another example, that of a stateful entity $\mathsf{rec}\, X.\, (\mathtt{off}?\,.\,\mathtt{on}?\,.\,X + M^{\ell} . X)$, emerges if you allow the enzyme to be (often temporarily) inhibited. More generally, the recursive process may be used to model phenomena such as recycling, replication, or the unbounded supplies of, e.g., nutrients associated with open systems. Note that recursion is often defined through a set of mutually recursive parametrised process constant definitions, $A(\boldsymbol{x}) \triangleq P$, and our definition may seem less elegant on first view. However, general recursion turns seemingly simple syntactic properties of processes, like *free names*, into fixpoint properties [24]. The present formulation of recursion allows us to capture each of the required fixpoints by a simple recursive definition, which will be of great value in later sections. We shall occasionally use equations of the form $\mathsf{A} \triangleq P$ to establish convenient abbreviations, but this should not be regarded as recursion.

The *parallel composition*, $P \mid Q$, denotes the concurrent composition of processes P and Q. In the resulting *reactive system* reactions may occur between P and Q if the one exposes a primitive, e.g. the capability enter n, and the other the *complementary* primitive, i.e. the (co-)capability accept n. If P and Q are themselves reactive systems, reactions may occur in each of them independently of the other. This corresponds to a chemical solution of reactive entities.

The *name restriction*, $(n)\,P$, restricts the scope of the name n to P. Technically, we may think of the name n as *bound* to some channel and the knowledge of the particular association being private to the sub-system P. Thus, two concurrent processes within P may react on n, but no process in P can react on n with a process outside of P. Biologically, this corresponds to a



(a) enter (b) exit (c) merge

Fig. 5. Movement styles of the BioAmbients calculus.

notion of confinement. However, the correspondence is not crystal-clear and we reserve the construct for the modelling of transient *coordination compounds*, i.e. complexes that temporarily come into existence due to the interaction of, e.g., enzymes and substrates.

The *ambient boundary* construct, $\lceil P \rceil^{\mu}$, denotes a process, P, encapsulated by a spatial boundary. There is an obvious correspondence to the biological concept of a *membrane bound compartment*, where a lipid bi-layer upholds a strong spatial separation between inside and outside fluids. It is also useful for the modelling of *molecular compartments*, where one or more molecules behave, e.g. fold or complexate, in a manner that effectively hides active domains or entire molecules from the environment. The language originally cast ambients as nameless entities, but in order to make sense of models and be able to track interesting information we shall ascribe a *role*, $\mu \in$ **Role**, to each ambient. Like the labels, roles have no semantic significance but are useful when designing our analyses in Sections 3 and 4.

Capability primitives. The effect of a reaction between processes is determined by the primitive prefixes involved. In the case of *capability primitives* the effect of reaction is that the local hierarchy of nested ambients changes.

The interaction of enter x and accept x allows an ambient to enter a neighbouring one (Fig. 5a). In the context of an ambient-as-membrane-bound-compartment abstraction this corresponds, e.g., to *endocytosis*, where a compartment (selectively) subsumes another large entity. In the context of an ambient-as-molecular-compartment abstraction it may correspond to a notion of (easily reversible) complexation. In a mixed context it may be used to describe proteins that migrate across compartment boundaries.

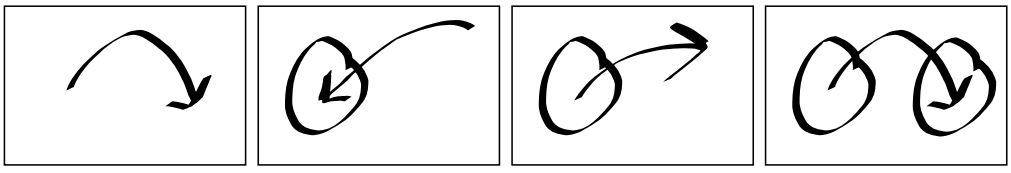
The interaction of exit x and expel x causes an ambient to leave the immediately enclosing one (Fig. 5b). In the context of an ambient-as-membrane-bound-compartment abstraction this may correspond to *exocytosis*, where a compartment (selectively) secretes some matter. For a ambient-as-molecular-compartment abstraction it corresponds, e.g., to the breaking of a complex. And again, in a mixed context it may describe proteins that migrate across compartment boundaries.

Finally, the interaction of merge+ x and merge-- x allows two neighbouring ambients to merge (Fig. 5c). In terms of ambient-as-membrane-bound-compartment abstractions this corresponds to fusion of biological compartments, whereas for ambient-as-molecular-compartment abstractions it corresponds to chemical bond formation or a notion of (hardly reversible) complexation. In a mixed context it may correspond to a virus (or drug delivery mechanism) that punches a hole in a membrane to deliver a payload.

Action primitives. In the case of *action primitives* the effect of reaction is information transfer incurring local changes, i.e. substitution, in the receiving process.

The interaction of a *sender*, $x!\{y\}$, and a *receiver*, $x?\{p\}$, allows *local communication*, where a process communicates a message to another process within the same ambient (Fig. 6a). In the context of an ambient-as-membrane-bound-compartment abstraction it may, e.g., correspond to enzyme-substrate reactions or, in conjunction with name restriction, the formation of transient coordination compounds. In the context of an ambient-as-molecular-compartment abstraction it may correspond to both intra-molecular and intra-complex interactions.

The interaction of sender $x\_!\{y\}$ and receiver $x\hat{}?\{p\}$ causes *parent to child communication*, i.e. a process may communicate a message to a process encapsulated by a neighbouring ambient (Fig. 6b). In the context of an ambient-as-membrane-bound-compartment abstraction this may correspond to receptor mediated interactions where the receiver models a surface receptor of a compartment. In an ambient-as-molecular-compartment abstraction it corresponds, e.g., to the interactions between simple molecules and complexes.



(a) local (b) parent to child (c) child to parent (d) sibling

Fig. 6. Communication styles of the BioAmbients calculus.

$$\begin{aligned}
\mathsf{fn}_\Gamma((n)\,P) &= \mathsf{fn}_\Gamma(P) \setminus \{n\} \\
\mathsf{fn}_\Gamma(\llbracket P \rrbracket^\mu) &= \mathsf{fn}_\Gamma(P) \\
\mathsf{fn}_\Gamma(P \mid Q) &= \mathsf{fn}_\Gamma(P) \cup \mathsf{fn}_\Gamma(Q) \\
\mathsf{fn}_\Gamma(\sum_{i\in I} M_i^{\ell_i}\,.\,P_i) &= \bigcup_{i\in I}(\mathsf{fn}(M_i) \cup \mathsf{fn}_\Gamma(P_i\ ) \setminus \mathsf{bn}(M_i)) \\
\mathsf{fn}_\Gamma(\mathsf{rec}\,X.\,P) &= \mathsf{fn}_{\Gamma\lfloor X\mapsto\emptyset\rfloor}(P) \\
\mathsf{fn}_\Gamma(X) &= \Gamma(X)
\end{aligned}$$

Fig. 7. Free names, $\mathsf{fn}_\Gamma(P)$, of processes, P.

In contrast, the interaction of sender $x\hat{}!\{y\}$ and receiver $x\_?\{p\}$ causes *child to parent communication* – a process may communicate a message to a process that is a neighbour of its immediately enclosing ambient (Fig. 6c). The biological correspondences are basically the same as in parent to child communication but with the directions reversed, i.e. the sender is a receptor.

Finally, the interaction of sender $x\#!\{y\}$ and receiver $x\#?\{p\}$ allows *sibling to sibling communication*, that is a process within some ambient may communicate a message to a process located within an ambient neighbouring the first (Fig. 6d). In the context of an ambient-as-membrane-bound-compartment this corresponds, e.g., to inter-compartment (e.g., hormonal) signalling. In an ambient-as-molecular-compartment setting it corresponds to inter-molecular or inter-complex interactions.

Canonical names. As custom for process calculi the operational semantics (Section 2.2) relies on α-renaming, i.e. the free replacement of a subterm $(n)\,Q$ of P by $(m)\,Q[^m/_n]$ whenever $m \notin \mathsf{fn}(Q)$. This is necessary to avoid capture when performing substitutions. Such two sub-terms are *α-equivalent*, written $(n)\,Q \equiv_\alpha (m)\,Q[^m/_n]$, and the semantic relations allow them to be freely exchanged. As a result neither n nor m are stable names for the underlying interaction mechanism.

The analyses presented in Sections 3, 4, and 5, however, are static, i.e. they compute the specified information by inspection of the initial syntax rather than exploration of the semantic configurations. For such information to constitute a safe over-approximation it must be valid for *all* semantic configurations and, hence, be based on stable information only. Therefore we associate each name, x, with a *canonical name*, $\lfloor x \rfloor$, and enforce *disciplined α-renaming*, i.e. the free replacement of a subterm $(n)\,Q$ of P by $(m)\,Q[^m/_n]$ whenever $m \notin \mathsf{fn}(Q)$ *and* $\lfloor n \rfloor = \lfloor m \rfloor$. In this manner the canonical names are preserved – even when the ordinary syntactical representations change.

When N is a set of names we shall write $\lfloor N \rfloor$ to denote the point-wise extension of $\lfloor x \rfloor$. Furthermore, when P is a process we shall write $\lfloor P \rfloor$ to denote the homomorphic extension of $\lfloor x \rfloor$. Finally, we assume the set of constants, $\mathbf{C}$, to be closed under canonicalisation, in the sense that $n \in \mathbf{C} \wedge \lfloor n \rfloor = \lfloor m \rfloor \Rightarrow m \in \mathbf{C}$.

Programs are processes P that satisfy the predicate $\mathsf{PRG}_\mathbf{C}(P)$ defined as the conjunction of the following well-formedness conditions (explained further below):

- P has no free process identifiers: $\mathsf{fpi}(P) = \emptyset$.
- P has free names only from the constants: $\mathsf{fn}(P) \subseteq \mathbf{C}$.
- P is well-formed with respect to the constants: $\mathbf{C} \vdash P$ (see Fig. 8 and below).

Here we write $\mathsf{fn}(P)$, which is shorthand for $\mathsf{fn}_{[\,]}(P)$ where $\mathsf{fn}_\Gamma(P)$ is defined in Fig. 7, for the *free names* of P and $\mathsf{fpi}(P)$ for the *free process identifiers* of P. Note that the former definition is parametrised by an environment, Γ, that constitutes a mapping from process identifiers to sets of free names, while the latter is not. This is required because sometimes, e.g. in Fig. 8, we shall have to compute the free names of sub-processes that are not closed with respect to process identifiers – meaning that further names may arise due to unfolding of recursion.

$$\text{WF-RES}\ \frac{\mathbf{C} \vdash_\Gamma P}{\mathbf{C} \vdash_\Gamma (n)\,P}\ \text{if}\ \lfloor n \rfloor \in \mathbf{C} \qquad \text{WF-AMB}\ \frac{\mathbf{C} \vdash_\Gamma P}{\mathbf{C} \vdash_\Gamma \llbracket P \rrbracket^\mu}\ \text{if}\ \mathsf{fpi}(P) = \emptyset$$

$$\text{WF-PAR}\ \frac{\mathbf{C} \vdash_\Gamma P \qquad \mathbf{C} \vdash_\Gamma P}{\mathbf{C} \vdash_\Gamma P \mid Q} \qquad \text{WF-SUM}\ \frac{\forall i \in I : \mathbf{C} \vdash_\Gamma P_i}{\mathbf{C} \vdash_\Gamma \sum_{i\in I} M_i^{\ell_i}\,.\,P_i}\ \text{if}\ \begin{array}{l}\forall i \in I : \\ (\lfloor \mathsf{bn}(M_i) \rfloor \cap \mathbf{C}) = \emptyset\end{array}$$

$$\text{WF-REC}\ \frac{\mathbf{C} \vdash_{\Gamma'} P}{\mathbf{C} \vdash_\Gamma \mathsf{rec}\,X.\,P}\ \text{if}\ \begin{array}{l} X \in \mathsf{fpi}(P) \\ \wedge\ \Gamma' = \Gamma[X \mapsto \mathsf{fn}_{\Gamma\lfloor X\mapsto\emptyset\rfloor}(P)] \\ \wedge\ \mathsf{nocap}_\Gamma\end{array} \qquad \text{WF-PID}\ \ \mathbf{C} \vdash_\Gamma X$$

where we write nocap_Γ for

$$(\forall (M^\ell\,.\,P') \preceq P : X \in \mathsf{fpi}(P') \Rightarrow \lfloor \mathsf{bn}(M) \rfloor \cap \lfloor \mathsf{fn}_\Gamma(\mathsf{rec}\,X.\,P) \rfloor = \emptyset)$$

Fig. 8. Well-formedness, $\mathbf{C} \vdash_\Gamma P$, of a process, P, with respect to a set of constants, $\mathbf{C}$.

The associated *well-formedness predicate*, $\mathbf{C} \vdash P$, which is shorthand for $\mathbf{C} \vdash_{[]} P$, enforces the implicit typing requirements imposed by the distinction between constants (in $\mathbf{C}$) and variables (in $\mathbf{V}$) in **Name**. The underlying predicate, $\mathbf{C} \vdash_{\Gamma} P$, is defined in Fig. 8, where we write $\mathsf{bn}(M)$ to denote the *bound names* of a prefix M (e.g. $\mathsf{bn}(n?\{p\}) = \{p\}$ whereas $\mathsf{bn}(n!\{m\}) = \{\}$) and $P \preceq Q$ to say that P is a sub-process of Q. The rule WF-RES ensures that names bound by restrictions are indeed in $\mathbf{C}$. The rule WF-SUM ensures that names bound by prefixes are not in $\mathbf{C}$.

Well-formedness prevents infinite nesting of ambients. Technically, this kind of behaviour is hard to handle in a static analysis [32] and, since it has no biological relevance, we omit it from well-formed processes. Consequently, the rule WF-AMB demands that the P in $\lceil P \rceil^{\mu}$ is process identifier closed. A similar choice was made for Mobile Safe Ambients in [18].

Well-formedness disallows useless recursive definitions. When the body, P, of a recursive definition, $\mathsf{rec}\,X.\,P$, does not make direct use of the associated process identifier, X, i.e. $X \notin \mathsf{fpi}(P)$, then the definition is useless. Thus, the rule WF-REC demands that $X \in \mathsf{fpi}(P)$.

Finally, well-formedness helps to enforce static scope and prevents identifier and name capture in the structural congruence to be defined below. In the case of process identifiers this is ensured by the fact that only the top-most recursive process can be unfolded. In the case of constants we rely on disciplined α-conversion in order to prevent capture. In the case of variables, however, we use the well-formedness condition to ensure that a process, $\mathsf{rec}\,X.\,P$, is only allowed to recur through a name binder, as in $P = \cdots . n?\{p\}^{\ell} . \cdots . X$, if P has no free occurrence of the bound name, i.e. p. This is ensured by the nocap part of rule WF-REC and prevents situations such as

$$n?\{p\}^{\ell} . \cdots . \mathsf{rec}\,X. \cdots p \cdots . n?\{p\}^{\ell} . \cdots . X$$

and

$$n?\{p\}^{\ell} . \cdots . \mathsf{rec}\,Y. \cdots p \cdots . \mathsf{rec}\,X. \cdots Y \,\big|\, n?\{p\}^{\ell} . \cdots . X.$$

Initial programs. Programs, $P_{\bigstar}$, that satisfy both $\mathsf{PRG}_{\mathbf{C}}(P_{\bigstar})$ and $P_{\bigstar} = \lfloor P_{\bigstar} \rfloor$, we call *initial*. Initial programs shall be our primary objects of interest in the following sections.

Syntactic Conventions. As customary for process calculi we shall abstain from writing the terminal $\mathbf{0}$ at the end of processes when writing examples. Similarly, we shall omit trivial name restrictions and simply *assume that* all free entities of a specification are constants and that all processes considered are parts of well-formed programs.

Example 1. Our running example is the following program P_{eat} (see line 6), which is inspired by the production of metabolites by catabolism of nutrients; as we shall explain later it models how 'food particles' (carriers of nutrients) may either be filtered out of the blood stream (lines 1 and 2) or subsumed by the cell (lines 2 and 5), possibly digested (lines 2, 5, and 3), and emitted as secretion (lines 1, 2 and 5):

let

(1) $\mathsf{Filter} \triangleq \mathsf{rec}\,Z.\,\mathsf{expel}\ rj^{1}\,.\,Z$

(2) $\mathsf{Transport} \triangleq \mathsf{rec}\,Y.\,rea\hat{}?\{rl\}^{2}\,.\,\mathsf{expel}\ rl^{3}\,.\,Y + \mathsf{exit}\ rj^{4}\,.\,Y + \mathsf{enter}\ ac^{5}\,.\,Y$

(3) $\mathsf{Nutrient} \triangleq \lceil \mathsf{exit}\ RL^{6} \rceil\,[nutrient]$

(4) $\mathsf{Food} \triangleq \lceil \mathsf{Transport} \,|\, \mathsf{Nutrient} \rceil\,[food]$

(5) $\mathsf{Cell} \triangleq \lceil \mathsf{rec}\,S.\,rea\_!\{RL\}^{7}\,.\,S + \mathsf{expel}\ rj^{8}\,.\,S + \mathsf{accept}\ ac^{9}\,.\,S \rceil\,[cell]$

in

(6) $\lceil \mathsf{Filter} \,|\, \mathsf{Food} \,|\, \mathsf{Cell} \rceil\,[system]$

The well-formedness condition is satisfied if we take $\mathbf{C} = \{rj, ac, rea, RL\}$.

2.2. Semantics

In the following we shall define the semantics of processes in BioAmbients. When conceiving of BioAmbients, Regev [39,38] gave the language a reaction semantics in the style of Berry and Boudol's Chemical Abstract Machine [2], which is the traditional choice for ambient calculi. As argued above this is a natural choice because it ensures a high degree of coherence between the inherently bio-chemical modelling domain and the operational model of the language. It is custom to define a reaction semantics in terms of *structural congruence*, $\equiv$, and *reaction*, $\longrightarrow$, both binary relations on processes. We shall diverge slightly from this tradition and, following Berry and Boudol's original proposal rather closely, define the semantics in terms of *heating*, $\Rrightarrow$, and reaction, $\xrightarrow{\tilde{\ell}}$.

Scope of restrictions:

H-SRES $(n)\,(m)\,P \equiv (m)\,(n)\,P$

H-SAMB $(n)\,(\lfloor P \rfloor^{\mu}) \equiv \lfloor (n)\,P \rfloor^{\mu}$

H-SEXT $(n)\,(P \mid Q) \equiv ((n)\,P) \mid Q$ if $n \notin \mathsf{fn}(Q)$

α-equivalence:

H-ALPH $(n)\,P \equiv (m)\,(P[{}^{m}/_{n}])$ if $m \notin \mathsf{fn}(P)$ and $\lfloor n \rfloor = \lfloor m \rfloor$

Evaporation and diffusion:

H-NEVA $P \mid \mathbf{0} \Rrightarrow P$

H-RDIF $(n)\,\mathbf{0} \Rrightarrow \mathbf{0}$

Reordering of parallel processes:

H-PCOM $P \mid Q \equiv Q \mid P$

H-PASS $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$

Reordering of summands:

H-SCOM $P + Q \equiv Q + P$

H-SASS $(P + Q) + R \equiv P + (Q + R)$

Unfolding of recursive processes:

H-UREC $\mathsf{rec}\,X.\,P \Rrightarrow P[{}^{\mathsf{rec}\,X.\,P}/_{X}]$

Reflexivity and transitivity:

H-REFL $P \Rrightarrow P$

H-TRAN $\dfrac{P \Rrightarrow Q \qquad Q \Rrightarrow R}{P \Rrightarrow R}$

Congruence:

H-CRES $\dfrac{P \Rrightarrow Q}{(n)\,P \Rrightarrow (n)\,Q}$

H-CAMB $\dfrac{P \Rrightarrow Q}{\lfloor P \rfloor^{\mu} \Rrightarrow \lfloor Q \rfloor^{\mu}}$

H-CPAR $\dfrac{P \Rrightarrow Q}{P \mid R \Rrightarrow Q \mid R}$

H-CSUM $\dfrac{P \Rrightarrow Q}{M^{\ell}.\,P + R \Rrightarrow M^{\ell}.\,Q + R}$

Fig. 9. The heating relation $P \Rrightarrow Q$ on processes. We write $P \equiv Q$ when both $P \Rrightarrow Q$ and $P \Lleftarrow Q$.

Heating relation. The *heating relation*, $\Rrightarrow$, is the least binary relation on **Proc** that is inductively defined by the axioms and rules of table in Fig. 9. When it holds between two process expressions P and Q, written $P \Rrightarrow Q$, it means that Q arises from P by any number of occurrences of

- insignificant syntactic restructuring (stirring, $\equiv$),
- elimination of an inactive process (evaporation, $\Rrightarrow$),
- elimination of a useless restriction (diffusion, $\Rrightarrow$), and
- unfolding of a recursive process (catalysis, $\Rrightarrow$).

Note that the ordinary structural congruence relation [38] is nearly fully embedded in the definition, hence we use $P \equiv Q$ as shorthand when both $P \Rrightarrow Q$ and $P \Lleftarrow Q$. However, we disallow the random introduction of inactive processes and vacuous restrictions, and assert that recursive processes can only be unfolded and not folded back.

As we shall see below two entities are able to react if they are (syntactically) close to each other in the sense that they are separated by a single $\mid$ and not separated by name restrictions. The first requirement is handled by the reordering rules, which make $\mid$ behave like a commutative monoid (with $\mathbf{0}$ as neutral element) and also give $+$ suitable reordering properties. The second requirement is addressed by the scope rules for restrictions, which ensure that obstructing restrictions can be migrated out of the way. In particular they allow restrictions to migrate in and out of parallel compositions.

We let the heating relation take care of the unfolding of recursive processes in the manner that is required for reaction semantics to allow reaction between two recursive processes. Here we write $P[Q/X]$ for the process that is as P except that all free occurrences of the process identifier, X, are replaced by the process expression Q. Clearly, such a notion of *substitution of free process identifiers* has the potential of causing *name* or *identifier capture*, i.e. that a free name or process identifier of Q is unintentionally redefined by a name restriction or recursion construct in P when substitution is performed. However, given that a substitution, $P[{}^{Q}/_{X}]$, always arises from the unfolding of the top-most recursive process, it is safe to assume that

$$
\begin{aligned}
((n)\,P)[^{Q}/_{X}] &= \begin{cases} (n)\,P[^{Q}/_{X}] & \text{if } n \notin \mathsf{fn}(Q) \\ (n')\,P[^{n'}/_{n}][^{Q}/_{X}] & \text{otherwise} \\ \quad \text{where } n' \notin (\mathsf{fn}(Q) \cup \mathsf{fn}(P)) \wedge \lfloor n' \rfloor = \lfloor n \rfloor & \end{cases} \\
(\llbracket\, P \,\rrbracket^{\mu})[^{Q}/_{X}] &= \llbracket\, P \,\rrbracket^{\mu} \\
(P_1 \mid P_2)[^{Q}/_{X}] &= P_1[^{Q}/_{X}] \mid P_2[^{Q}/_{X}] \\
(\sum_{i\in I} M_i^{\ell_i} . P_i)[^{Q}/_{X}] &= \sum_{i\in I} M_i^{\ell_i} . P_i[^{Q}/_{X}] \\
(\mathsf{rec}\, Y.\, P)[^{Q}/_{X}] &= \begin{cases} \mathsf{rec}\, Y.\, P[^{Q}/_{X}] & \text{if } X \neq Y \\ \mathsf{rec}\, Y.\, P & \text{otherwise} \end{cases} \\
Y[^{Q}/_{X}] &= \begin{cases} Q & \text{if X = Y} \\ Y & \text{otherwise} \end{cases}
\end{aligned}
$$

Fig. 10. Substitution, $P[^{Q}/_{X}]$, of a process Q for an identifier X in a process P.

- Q is the top-level recursive process that defines X, i.e., $Q = \mathsf{rec}\, X.\, Q'$,
- Q is identifier closed, i.e., $\mathsf{fpi}(Q) = \emptyset$,
- Q is a sub-process of a well-formed program, and hence $\mathbf{C} \vdash Q$, and
- P is a sub-process of Q, i.e., $P \prec Q$ or, equivalently, $P \preceq Q'$.

This justifies the particularly simple definition of substitution in Fig. 10. The substitution over restrictions is nearly the standard one, i.e., constant capture is avoided by α-renaming. The capture of variables is not possible because of the separation of **Name** into **C** and **V**. Similarly the substitution over summations is correctly defined because $\mathsf{bn}(M_i) \cap \mathsf{fn}_{\Gamma}(Q) = \emptyset$ when $X \in \mathsf{fpi}(P)$. The substitution over ambients is correctly defined because $\mathsf{fpi}(P) = \emptyset$. The substitution over recursive processes is correctly defined because $Y \notin \mathsf{fpi}(Q)$ due to $\mathsf{fpi}(Q) = \emptyset$. Finally, the substitution over process identifiers is straightforward to define.

The heating relation enforces α-*equivalence*, i.e. that processes are identical if they differ only in their choice of bound names (subject to our consideration of disciplined α-renaming). The separation of constants and variables ensures that we never substitute one variable for another and, therefore, we need α-equivalence only for constants. However, α-equivalence, and the substitution of identifiers, rely on the notion of *substitution of free names*, i.e. writing $P[^{m}/_{x}]$ to denote the process that is as P except that every free occurrence of x is replaced by m. This notion of substitution is defined in Fig. 11. We rely on α-renaming to avoid name capture when substituting over name restriction. Furthermore, we define substitution into

$$
\begin{aligned}
((n)\,P)[^{m}/_{x}] &= \begin{cases} (n)\,P[^{m}/_{x}] & \text{if } n \neq x \wedge m \neq n \\ (n')\,P[^{n'}/_{n}][^{m}/_{x}] & \text{if } n \neq x \wedge m = n \\ \quad \text{where } n' \notin (\{n, m\} \cup \mathsf{fn}(P)) \wedge \lfloor n' \rfloor = \lfloor n \rfloor & \\ (n)\,P & \text{otherwise} \end{cases} \\
(\llbracket\, P \,\rrbracket^{\mu})[^{m}/_{x}] &= \llbracket\, P[^{m}/_{x}] \,\rrbracket^{\mu} \\
(P_1 \mid P_2)[^{m}/_{x}] &= P_1[^{m}/_{x}] \mid P_2[^{m}/_{x}] \\
(\sum_{i\in I} M_i^{\ell_i} . P_i)[^{m}/_{x}] &= \sum_{i\in I} M_i^{\ell_i}[^{m}/_{x}] . P_i' \qquad \text{where } P_i' = \begin{cases} P_i[^{m}/_{x}] & \text{if } x \notin \mathsf{bn}(M_i) \\ P_i & \text{otherwise} \end{cases} \\
(\mathsf{rec}\, X.\, P)[^{m}/_{x}] &= \mathsf{rec}\, X.\, P[^{m}/_{x}] \\
X[^{m}/_{x}] &= X
\end{aligned}
$$

Fig. 11. Substitution, $P[^{m}/_{x}]$, of a constant m for a name x in a process P.

Movement reactions:

R-ENT

$$\lfloor\, (\mathsf{enter}\ n^{\ell_1}.P + P')\ \big|\ P''\,\rfloor^{\mu_1}\ \big|\lfloor\, (\mathsf{accept}\ n^{\ell_2}.Q + Q')\ \big|\ Q''\,\rfloor^{\mu_2} \xrightarrow{(\ell_1,\ell_2)} \lfloor\lfloor\, P\ \big|\ P''\,\rfloor^{\mu_1}\ \big|\ Q\ \big|\ Q''\,\rfloor^{\mu_2}$$

R-EXT

$$\lfloor\lfloor\, (\mathsf{exit}\ n^{\ell_1}.P + P')\ \big|\ P''\,\rfloor^{\mu_1}\ \big|\ (\mathsf{expel}\ n^{\ell_2}.Q + Q')\ \big|\ Q''\,\rfloor^{\mu_2} \xrightarrow{(\ell_1,\ell_2)} \lfloor\, P\ \big|\ P''\,\rfloor^{\mu_1}\ \big|\lfloor\, Q\ \big|\ Q''\,\rfloor^{\mu_2}$$

R-MRG

$$\lfloor\, (\mathsf{merge}\text{–}\ n^{\ell_1}.P + P')\ \big|\ P''\,\rfloor^{\mu_1}\ \big|\lfloor\, (\mathsf{merge}\text{+}\ n^{\ell_2}.Q + Q')\ \big|\ Q''\,\rfloor^{\mu_2} \xrightarrow{(\ell_1,\ell_2)} \lfloor\, P\ \big|\ P''\ \big|\ Q\ \big|\ Q''\,\rfloor^{\mu_2}$$

Communication reactions:

R-L2L

$$(n!\{m\}^{\ell_1}.P + P')\ \big|\ (n?\{p\}^{\ell_2}.Q + Q') \xrightarrow{(\ell_1,\ell_2)} P\ \big|\ Q[m/p]$$

R-P2C

$$(n\_!\{m\}^{\ell_1}.P + P')\ \big|\lfloor\, (n\hat{}?\{p\}^{\ell_2}.Q + Q')\ \big|\ Q''\,\rfloor^{\mu} \xrightarrow{(\ell_1,\ell_2)} P\ \big|\lfloor\, Q[m/p]\ \big|\ Q''\,\rfloor^{\mu}$$

R-C2P

$$\lfloor\, (n\hat{}!\{m\}^{\ell_1}.P + P')\ \big|\ P''\,\rfloor^{\mu}\ \big|\ (n\_?\{p\}^{\ell_2}.Q + Q') \xrightarrow{(\ell_1,\ell_2)} \lfloor\, P\ \big|\ P''\,\rfloor^{\mu}\ \big|\ Q[m/p]$$

R-S2S

$$\lfloor\, (n\#!\{m\}^{\ell_1}.P + P')\ \big|\ P''\,\rfloor^{\mu_1}\ \big|\lfloor\, (n\#?\{p\}^{\ell_2}.Q + Q')\ \big|\ Q''\,\rfloor^{\mu_2} \xrightarrow{(\ell_1,\ell_2)} \lfloor\, P\ \big|\ P''\,\rfloor^{\mu_1}\ \big|\lfloor\, Q[m/p]\ \big|\ Q''\,\rfloor^{\mu_2}$$

Reaction in context:

$$\text{R-RES}\ \frac{P \xrightarrow{\tilde{\ell}} Q}{(n)\,P \xrightarrow{\tilde{\ell}} (n)\,Q} \qquad \text{R-AMB}\ \frac{P \xrightarrow{\tilde{\ell}} Q}{\lfloor\, P\,\rfloor^{\mu} \xrightarrow{\tilde{\ell}} \lfloor\, Q\,\rfloor^{\mu}} \qquad \text{R-PAR}\ \frac{P \xrightarrow{\tilde{\ell}} Q}{P\ \big|\ R \xrightarrow{\tilde{\ell}} Q\ \big|\ R}$$

Heating:

$$\text{R-AUX}\ \frac{P \Rrightarrow P' \quad P' \xrightarrow{\tilde{\ell}} Q' \quad Q' \Rrightarrow Q}{P \xrightarrow{\tilde{\ell}} Q}$$

Fig. 12. The reaction relation, $P \xrightarrow{\tilde{\ell}} Q$, on processes.

prefixes $M_i^{\ell_i}[^m/_x]$ such that defining, or *bound*, occurrences of variables (i.e. $\mathsf{bn}(M_i^{\ell_i})$) are *not* subject to substitution. The substitution over recursive processes is correctly defined. Due to the well-formedness of P we are ensured that, in case x is a variable, then x occurs free in P only if this cannot lead to capture. If x is a constant we rely on α-renaming to avoid capture.

The reaction relation is the least relation, $\xrightarrow{\tilde{\ell}}$ (where $\tilde{\ell}$ is a pair of labels), defined inductively by the axioms and rules of Fig. 12 and it constitutes our definition of the reactive behaviour of processes. When it holds between two processes P and Q, written $P \xrightarrow{(\ell_1,\ell_2)} Q$, it means that P can evolve into Q by a single movement or communication reaction involving two prefixes that are labelled ℓ_1 and ℓ_2, respectively. In the sequel we shall write $P_\star \xrightarrow{\tilde{L}} {}^\star P \xrightarrow{\tilde{\ell}} Q$ for a sequence of reactions that evolve an initial program $P_\star$, first into P via the sequence $\tilde{L}$ of reactions, and then into Q via a final reaction $\tilde{\ell}$.

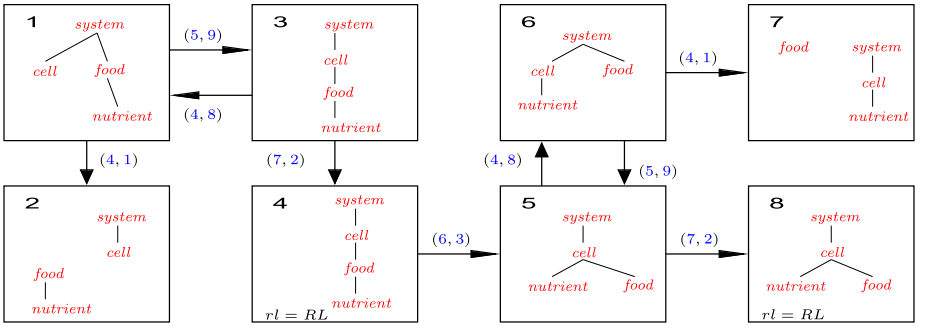
The labelling of the relation constitutes an *instrumentation* and we shall later depend on it in order to prove properties of the defined analyses. We stress that the labels do not constrain the reaction relation in any way but are propagated for bookkeeping purposes.

The fundamental semantic event is binary *reaction*. Reaction is possible when two constituents of a certain characteristic form (e.g. $\lfloor\, \big|_i \sum_{j_i} M_{j_i}^{\ell_{j_i}}.P_{j_i}\,\rfloor^{[\mu]}$ for movement and $\big|_i \sum_{j_i} M_{j_i}^{\ell_{j_i}}.P_{j_i}$ for communication) expose complementary prefixes and occur in a suitable parallel configuration. Due to *reaction in context*, reaction can happen anywhere in a system, and *heating* ensures that insignificant deviations from the prescribed forms do not prevent reaction.

Every reaction has two effects. In the case of movement the local ambient hierarchy is changed and in the case of communication a constant is substituted for a variable in the receiving process. In both cases the prefixes involved in the synchronising reaction are removed to leave room for their continuations and any summands are discarded.

Note that the more traditional style of structural operational semantics regards unary *action* as the fundamental event and must recover and match two actions from sub-systems in order to infer a binary reaction.

Example 2. The semantics of the example program P_{eat} is illustrated below:



The initial configuration is shown in frame 1 and here the tree structure reflects a scenario where *cell* and *food* are siblings inside *system* and *nutrient* is a sub-ambient of *food*. In this configuration (4,1) can react to move *food* out of *system* and obtain the stuck configuration of frame 2. Alternatively, (5,9) can react to move *food* into *cell* (frame 3). Then (4,8) can react to move *food* back out of *cell* (frame 1 again), or (7,2) can react to bind the variable *rl* to the constant *RL* (frame 4). After that only (6,3) can react to move *nutrient* out of *food* (frame 5). Here (7,2) may react to bind the variable *rl* to the constant *RL* once more and thereby obtain the stuck configuration of frame 8. Alternatively, (4,8) can react to move *food* out of *cell* (frame 6). From here (5,9) may react to move *food* back into *cell* (frame 5). Alternatively, (4,1) can move *food* out of *system* and thereby obtain the stuck configuration of frame 7.

2.3. Properties of programs

Later, when proving properties of the analysis to be defined in Sections 3, 4, and 5, we shall rely on the well-formedness conditions of programs. We can do this because programs evaluate into programs:

Lemma 1 (Type soundness). *Assume that* $P_\star$ *is a well-formed initial program then* $P_\star \xrightarrow{\tilde{L}}{}^\star P \xrightarrow{\tilde{\ell}} Q$ *implies that* $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $\mathsf{PRG}_{\mathbf{C}}(Q)$.

Proof. The result follows by induction of the length of the derivation sequence $\tilde{L}$, where we use Fact 26 from Appendix A to establish the base case and, in conjunction with the induction hypothesis, to establish the inductive step. □

Note that this result is particular to the set of well-formedness conditions put forward in this paper. The original proposal of BioAmbients [38,39] never formalised such properties.

3. Control flow analysis

When we define our pathway analysis in Section 4 we shall find it useful to have at our disposal a reasonably precise estimate of the reactions that may take place in the modelled system. In this section we will show how a fairly mundane static control flow analysis (CFA) in the style of [25,32] can be used to obtain such an estimate.

Control Flow Analysis problems emerge when one tries to compute how focus of control moves through a program. The initial conceptualisation of static Control Flow Analysis dates back to work on interprocedural analysis and obtained momentum with Shivers' work on functional languages [41], and was further refined by Jagannathan [13]. In this context, and based on the view that, in terms of flow, data and control are two sides of the same thing [23], *Flow Logic* was pioneered in the late 1990s, by Nielson and Nielson [29,28,22], as a unifying specification oriented approach to constraint based static analysis.

The Flow Logic framework makes a clear distinction between the specification of an analysis and the computation of corresponding analysis results. This approach allows the designer to focus on the specification of analyses without making compromises dictated by implementation considerations. The implementation phase is also simplified and improved, as the implementer is always free to choose the best available tool – no particular tool or formalism is prescribed by the framework.

3.1. The analysis domain

As usual for static analyses a Flow Logic specification is based on a suitable universe of discourse, $\mathfrak{A}$, usually called the *analysis domain*. It is customary to demand that $\mathfrak{A}$ is a complete lattice designed such that each element, $\mathcal{A}$, corresponds to *global* Control Flow Analysis information of the program of interest.

In our case the analysis specifies the following three components:

- An approximation of the relevant name bindings:

  $$\mathscr{R} \subseteq \mathbf{V} \times \mathbf{C}$$

  where we write $n \in \mathscr{R}(p)$ or $(p,n) \in \mathscr{R}$ to assert the truth of predicate $\mathscr{R}(p,n)$, i.e. that $\mathscr{R}$ records that the variable p may be bound to the name n.
- An approximation of the contents (ambients, prefixes) of ambients:

  $$\mathscr{I} \subseteq \mathbf{Role} \times (\mathbf{Role} \cup (\mathbf{CA} \times \mathbf{Lab}))$$

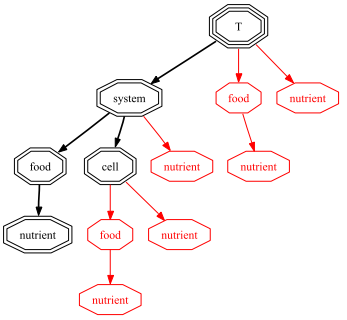
  where we write $\mu \in \mathscr{I}(\mu')$ or $(\mu',\mu) \in \mathscr{I}$ (or, $M^{\ell} \in \mathscr{I}(\mu')$ or $(\mu',M^{\ell}) \in \mathscr{I}$) to denote the truth of predicate $\mathscr{I}(\mu',\mu)$ (or $\mathscr{I}(\mu',M^{\ell})$), i.e. that $\mathscr{I}$ records that an ambient of role μ (or a prefix M^{ℓ}) may occur inside an ambient of role μ'.
- An approximation of the pairs of capabilities that may react:

  $$\mathscr{F} \subseteq \mathbf{Lab} \times \mathbf{Lab}$$

  where we shall write $(\ell_1,\ell_2) \in \mathscr{F}$ to denote the truth of $\mathscr{F}(\ell_1,\ell_2)$, i.e. that $\mathscr{F}$ records that prefixes labelled ℓ_1 and ℓ_2 may react.

The domain, $\mathfrak{A}$, of our CFA is given as the Cartesian product of those corresponding to the three components, such that $\mathcal{A} = (\mathscr{I},\mathscr{R},\mathscr{F})$ is a typical element. This clearly constitutes a complete lattice under the component-wise subset ordering.

Example 3. The best analysis estimate $(\mathscr{I}_{\mathsf{eat}},\mathscr{R}_{\mathsf{eat}},\mathscr{F}_{\mathsf{eat}})$ of the running example P_{eat} is shown below:



p	$\mathcal{R}_{\mathsf{eat}}(p)$
rl	RL

c	$\mathcal{F}_{\mathsf{eat}}(c)$
exit rj^4	expel rj^1, expel rj^8
enter ac^5	accept ac^9
exit RL^6	expel rl^3
$rea\_!\{RL\}^7$	$rea\hat{}?\{rl\}^2$

In the figure the contents of $\mathscr{I}_{\mathsf{eat}}$ is presented graphically, while $\mathscr{R}_{\mathsf{eat}}$ and $\mathscr{F}_{\mathsf{eat}}$ are presented as tables where last components are sorted into bins identified by first components. In the graph the triple bordered node represents the super-environment used as superscript in $(\mathscr{I}_{\mathsf{eat}},\mathscr{R}_{\mathsf{eat}},\mathscr{F}_{\mathsf{eat}}) \models^{\top} P_{\mathsf{eat}}$, the double bordered nodes connected by bold lines represent the initial configuration (as specified by Example 1), and the remaining nodes represent the the nestings of ambients that may be reachable through interactions, starting from the initial state. The trees of the individual frames of Example 2 are all sub-trees of this graph.

3.2. The acceptability judgement

Fundamentally, a Flow Logic specification is concerned with the relationship between programs $P \in \mathbf{Proc}$ and static analysis estimates $\mathcal{A} \in \mathfrak{A}$. This connection is captured by an acceptability judgement

$$\mathcal{A} \models P$$

intended to hold precisely when $\mathcal{A}$ constitutes an acceptable analysis estimate for P.

The judgement is defined by clauses; typically there is one clause for each syntactic construct ϕ of **Proc** and they take the form

$$\mathcal{A} \models \phi(\cdots P_i \cdots) \quad \text{iff} \quad \begin{array}{l}(\text{some formula } \varphi \text{ with } \mathcal{A} \models P' \\ \text{for various sub-programs } P')\end{array}$$

where φ is usually a formula of a suitable fragment of First Order Logic, FOL.

These ingredients formalise a given Control Flow Analysis problem as a *Flow Logic*. Obviously the involved judgement relation "$\models$" has functionality

$$\models : (\mathfrak{A} \times \mathbf{Proc}) \to \{\text{true},\text{false}\}$$

$$
\begin{array}{ll}
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} (n)\,P & \text{iff } (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P \\
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} \lfloor P \rfloor^{\mu_c} & \text{iff } \mu_c \in \mathcal{I}(\mu) \wedge (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu_c} P \\
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P \mid P' & \text{iff } (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P \wedge (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P' \\
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} \sum_{i\in I} M_i^{\ell_i}.P_i & \text{iff } \forall i \in I : (\lfloor M_i \rfloor^{\ell_i} \in \mathcal{I}(\mu) \wedge \mathsf{closure}_{\lceil M_i \rceil} \wedge (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P_i) \\
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} \mathsf{rec}\,X.\,P & \text{iff } (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P \\
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} X & \text{iff } \mathsf{true}
\end{array}
$$

Fig. 13. The 0CFA acceptability judgement.

In our case it can be defined in a syntax directed manner (and hence we dispense with the more general co-inductive definition).

For the present analysis the acceptability judgement takes the form

$$(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P$$

and expresses that, when the sub process P (of $P_\star$) is enclosed within an ambient of role $\mu \in \mathbf{Role}$, then $\mathscr{I}$, $\mathscr{R}$, and $\mathscr{F}$ correctly capture the behaviour of P – meaning that $\mathscr{I}$ approximates the contents that may occur in each ambient, $\mathscr{R}$ the bindings of names that may take place, and $\mathscr{F}$ the prefix pairs that may react, as P evolves inside $P_\star$. The judgement is specified in Fig. 13 and refers to Figs. 14 and 15 for specifications of the closure conditions $\mathsf{closure}_{\lceil M \rceil}$, where $\lceil M \rceil$ is as M but with names replaced by '·'. The judgement is defined by structural recursion over the syntax of processes (which makes our 0CFA specification *compositional* in the terminology of Flow Logic).

The resulting judgement asserts that an analysis estimate, $(\mathscr{I},\mathscr{R},\mathscr{F})$, is acceptable for a restricted process, $(n)\,P$, in the context of an ambient of role, μ, if and only if acceptable for P in μ. Avoiding further requirements helps us ensure that the analysis is invariant under structural congruence. Instead we defer the treatment of constant definitions to the auxiliary closure conditions, $\mathsf{closure}_{\lceil M \rceil}$, defined further below.

Regarding ambients we consider an analysis estimate, $(\mathscr{I},\mathscr{R},\mathscr{F})$, is acceptable for $\lfloor P \rfloor^{\mu_c}$ in μ if and only if $(\mu,\mu_c) \in \mathcal{I}$ and $(\mathscr{I},\mathscr{R},\mathscr{F})$ is acceptable for P in μ_c.

For guarded sums an estimate, $(\mathscr{I},\mathscr{R},\mathscr{F})$, is acceptable for $\sum_{i\in I} M_i^{\ell_i}.P_i$ in μ if and only if for every $i \in I$ it is the case that $(\mu,M_i^{\ell_i}) \in \mathcal{I}$, the associated closure condition $\mathsf{closure}_{\lceil M_i \rceil}$ holds for $(\mathscr{I},\mathscr{R},\mathscr{F})$, and $(\mathscr{I},\mathscr{R},\mathscr{F})$ is acceptable for P_i in μ.

Similarly, an analysis, $(\mathscr{I},\mathscr{R},\mathscr{F})$, is acceptable for $P \mid P'$ in μ simply if acceptable for both P and P' in μ. Note that this makes the acceptability conditions for parallel and choice appear identical. We leave it to the auxiliary closure conditions $\mathsf{closure}_{\lceil M \rceil}$ to ensure a differentiated treatment of the two types of composition.

In the case of recursion an analysis estimate, $(\mathscr{I},\mathscr{R},\mathscr{F})$, is acceptable for $\mathsf{rec}\,X.\,P$ in μ if acceptable for P in μ. In the case of process identifiers we effectively ignore them by accepting any $(\mathscr{I},\mathscr{R},\mathscr{F})$ as an analysis estimate for X in μ. This simple treatment of recursion is acceptable for two reasons: Firstly, the well-formedness condition ensures that no process identifier occurs free inside an ambient; thus, it suffices to analyse the sub-process P (the body) in the context where it is first defined. And, secondly, the CFA is *flow-insensitive*, i.e. it does not consider the actual sequencing of prefixes, and thus unfolding is not required.

3.3. Closure conditions

The *closure conditions*, $\mathsf{closure}_{\lceil M \rceil}$, defined in Figs. 14 and 15 are intended to mimic the reaction semantics. In each case the pre-conditions check whether an abstract version of the corresponding semantic reaction condition is fulfilled by $(\mathscr{I},\mathscr{R},\mathscr{F})$. This depends on the appropriate spatial placement of prefixes (as recorded by $\mathscr{I}$), agreement on bindings of names (as recorded by $\langle\mathscr{R}\rangle$ – see below), and whether the prefixes in question may be *concurrently possible* (as recorded by the auxiliary relation $\mathsf{CP}_\star$ – see Section 3.4). If the preconditions are satisfied by $(\mathscr{I},\mathscr{R},\mathscr{F})$ then the conclusion must also be satisfied by $(\mathscr{I},\mathscr{R},\mathscr{F})$ in order for $(\mathscr{I},\mathscr{R},\mathscr{F})$ to be an acceptable analysis estimate. In the case of capabilities we require that $\mathcal{I}$ appropriately reflects the spatial configurations that might arise from the corresponding reaction. In contrast, for actions we require that $\mathcal{R}$ appropriately reflects the name bindings that might arise from the corresponding reaction. In both cases we require that $\mathcal{F}$ duly records the pair of prefixes involved in the potential reaction.

As the relation $\mathscr{R}$ is only concerned with the bindings of variables to constants the closure conditions also refer to an extended version

$$\langle\mathscr{R}\rangle = (\{(n,n) \mid n \in \mathbf{C}\} \cup \mathscr{R}) \subseteq (\mathbf{V} \cup \mathbf{C}) \times \mathbf{C}$$

that takes care of variables as well as constants. Basically, it acts exactly like $\mathcal{R}$ with respect to variable bindings but also acts as the identity on constants, recording the binding of each constant to the constant itself.

3.4. Concurrently possible prefixes

In deciding the acceptability of the closure conditions we use the auxiliary relation $\mathsf{CP}_\star$, which is a precomputed approximation of the *concurrently possible* prefixes of the program $P_\star$ of interest. It is computed as $\mathsf{CP}_{[\,][\,]}(P_\star)$ using the recursive function $\mathsf{CP}_{\Gamma\Delta}(P)$ defined in Fig. 17. Basically, the computation is based on the simple observation that two prefixes have no chance of interacting if they are not syntactically located in parallel processes. Thus, in the case of parallel composition, the definition of $\mathsf{CP}_{\Gamma\Delta}(P)$ uses the auxiliary operation $\mathsf{cross}_\Gamma(P,Q)$ to record that prefixes in the two branches may interact with one another. The auxiliary function $\mathsf{prfx}_\Gamma(P)$, defined in Fig. 16, is used in order to specify this. In contrast, no such components are recorded for non-deterministic choice.

In these specifications the Γ index denotes a mapping from process identifiers to sets of prefix labels and the Δ index a mapping from process identifiers to sets of pairs of prefix labels. The mapping are necessary because sometimes (e.g. when $\mathsf{prfx}_\Gamma(P)$ is invoked from within $\mathsf{CP}_{\Gamma\Delta}(P)$) we shall need to collect information (e.g. the occurring prefixes) from sub-processes that have free processes identifiers – meaning that further information may arise from unfolding.

Clearly, it is only safe for the closure conditions to rely on $\mathsf{CP}_\star$ if $\mathsf{CP}_\star$ itself safely over-approximates the set of prefixes that may become concurrently possible at run-time:

Lemma 2 ($\mathsf{CP}_\star$ is semantically correct). *Assume that $P_\star$ is a well-formed initial program, then*

$$\text{if } P_\star \xrightarrow{\tilde{L}}{}^\star P \xrightarrow{\tilde{\ell}} Q \text{ then } \mathsf{CP}_\star \supseteq \mathsf{CP}(P) \supseteq \mathsf{CP}(Q).$$

Proof. The result follows by induction on the length of $\tilde{L}$. The base case follows from Corollary 31, and the inductive step follows from the induction hypothesis in conjunction with Lemma 1 and Corollary 31. □

3.5. Properties of the CFA

For such a setup to qualify as an actual Flow Logic it must posses a number of desirable properties that we shall explain in the following.

$$\begin{aligned}
\mathsf{closure}_{\mathsf{enter}}\ . \quad &= \forall \mu, \mu_1, \mu_2, x, y, \ell_1, \ell_2 : \\
&\quad \mathsf{enter}\ x^{\ell_1} \in \mathcal{I}(\mu_1) \wedge \mu_1 \in \mathcal{I}(\mu) \wedge \\
&\quad \mathsf{accept}\ y^{\ell_2} \in \mathcal{I}(\mu_2) \wedge \mu_2 \in \mathcal{I}(\mu) \wedge \\
&\quad \langle \mathcal{R} \rangle(x) \cap \langle \mathcal{R} \rangle(y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
&\quad \Rightarrow \quad \mu_1 \in \mathcal{I}(\mu_2) \wedge (\ell_1, \ell_2) \in \mathcal{F} \\
\mathsf{closure}_{\mathsf{accept}}\ . \quad &= \mathsf{true} \\
\mathsf{closure}_{\mathsf{exit}}\ . \quad &= \forall \mu, \mu_1, \mu_2, x, y, \ell_1, \ell_2 : \\
&\quad \mathsf{exit}\ x^{\ell_1} \in \mathcal{I}(\mu_1) \wedge \mu_1 \in \mathcal{I}(\mu_2) \wedge \\
&\quad \mathsf{expel}\ y^{\ell_2} \in \mathcal{I}(\mu_2) \wedge \mu_2 \in \mathcal{I}(\mu) \wedge \\
&\quad \langle \mathcal{R} \rangle(x) \cap \langle \mathcal{R} \rangle(y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
&\quad \Rightarrow \quad \mu_1 \in \mathcal{I}(\mu) \wedge (\ell_1, \ell_2) \in \mathcal{F} \\
\mathsf{closure}_{\mathsf{expel}}\ . \quad &= \mathsf{true} \\
\mathsf{closure}_{\mathsf{merge}-}\ . \quad &= \forall \mu, \mu_1, \mu_2, x, y, \ell_1, \ell_2 : \\
&\quad \mathsf{merge}- \ x^{\ell_1} \in \mathcal{I}(\mu_1) \wedge \mu_1 \in \mathcal{I}(\mu) \wedge \\
&\quad \mathsf{merge}+ \ y^{\ell_2} \in \mathcal{I}(\mu_2) \wedge \mu_2 \in \mathcal{I}(\mu) \wedge \\
&\quad \langle \mathcal{R} \rangle(x) \cap \langle \mathcal{R} \rangle(y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
&\quad \Rightarrow \quad \mathcal{I}(\mu_1) \subseteq \mathcal{I}(\mu_2) \wedge (\ell_1, \ell_2) \in \mathcal{F} \\
\mathsf{closure}_{\mathsf{merge}+}\ . \quad &= \mathsf{true}
\end{aligned}$$

Fig. 14. Closure conditions regarding movement capabilities.

$$
\begin{aligned}
\mathsf{closure}_{!\{\cdot\}} &= \forall \mu, x, y, z, p, \ell_1, \ell_2 : \\
&\quad x!\{z\}^{\ell_1} \in \mathcal{I}(\mu) \wedge y?\{p\}^{\ell_2} \in \mathcal{I}(\mu) \wedge \\
&\quad \langle\mathcal{R}\rangle(x) \cap \langle\mathcal{R}\rangle(y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
&\quad \Rightarrow \quad \langle\mathcal{R}\rangle(z) \subseteq \mathcal{R}(p) \wedge (\ell_1, \ell_2) \in \mathcal{F} \\
\mathsf{closure}_{?\{\cdot\}} &= \mathsf{true} \\
\mathsf{closure}_{\_!\{\cdot\}} &= \forall \mu, \mu_1, x, y, z, p, \ell_1, \ell_2 : \\
&\quad x\_!\{z\}^{\ell_1} \in \mathcal{I}(\mu) \wedge y\hat{}?\{p\}^{\ell_2} \in \mathcal{I}(\mu_1) \wedge \mu_1 \in \mathcal{I}(\mu) \wedge \\
&\quad \langle\mathcal{R}\rangle(x) \cap \langle\mathcal{R}\rangle(y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
&\quad \Rightarrow \quad \langle\mathcal{R}\rangle(z) \subseteq \mathcal{R}(p) \wedge (\ell_1, \ell_2) \in \mathcal{F} \\
\mathsf{closure}_{\hat{}?\{\cdot\}} &= \mathsf{true} \\
\mathsf{closure}_{\hat{}!\{\cdot\}} &= \forall \mu, \mu_1, x, y, z, p, \ell_1, \ell_2 : \\
&\quad x\hat{}!\{z\}^{\ell_1} \in \mathcal{I}(\mu_1) \wedge \mu_1 \in \mathcal{I}(\mu) \wedge y\_?\{p\}^{\ell_2} \in \mathcal{I}(\mu) \wedge \\
&\quad \langle\mathcal{R}\rangle(x) \cap \langle\mathcal{R}\rangle(y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
&\quad \Rightarrow \quad \langle\mathcal{R}\rangle(z) \subseteq \mathcal{R}(p) \wedge (\ell_1, \ell_2) \in \mathcal{F} \\
\mathsf{closure}_{\_?\{\cdot\}} &= \mathsf{true} \\
\mathsf{closure}_{\#!\{\cdot\}} &= \forall \mu, \mu_1, \mu_2, x, y, z, p, \ell_1, \ell_2 : \\
&\quad x\#!\{z\}^{\ell_1} \in \mathcal{I}(\mu_1) \wedge \mu_1 \in \mathcal{I}(\mu) \wedge y\#?\{p\}^{\ell_2} \in \mathcal{I}(\mu_2) \wedge \mu_2 \in \mathcal{I}(\mu) \wedge \\
&\quad \langle\mathcal{R}\rangle(x) \cap \langle\mathcal{R}\rangle(y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
&\quad \Rightarrow \quad \langle\mathcal{R}\rangle(z) \subseteq \mathcal{R}(p) \wedge (\ell_1, \ell_2) \in \mathcal{F} \\
\mathsf{closure}_{\#?\{\cdot\}} &= \mathsf{true}
\end{aligned}
$$

Fig. 15. Closure conditions regarding communication capabilities.

Well-definedness

The analysis must be well-defined, i.e., for every combination of $P \in \mathbf{Proc}$ and $\mathcal{A} \in \mathfrak{A}$, the acceptability of $\mathcal{A}$ as an analysis estimate for P is unambiguously defined. This amounts to showing that "$\models$", with functionality as outlined above, constitutes a total function. In our case this is immediate due to the syntax directed definition of "$\models$":

Fact 3 (*Well-defined*). The analysis judgement $(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P$ is well-defined, i.e. for every pair $(\mathcal{I},\mathcal{R},\mathcal{F})$ and P it unambiguously specifies whether $(\mathcal{I},\mathcal{R},\mathcal{F})$ is acceptable for P.

Semantic correctness

Flow Logic is a *semantics based* approach to static analysis, i.e. the analysis information can be proved *correct* with respect to a semantic specification. Intuitively, we want to show that acceptable estimates are safe approximations, i.e. acceptable for every reachable semantic configuration rather than just the initial one.

Formally, the analysis is *semantically correct* if the acceptability of estimates enjoys a *subject reduction* result:

$$\text{if } \mathcal{A} \models P \text{ and } P \xrightarrow{\tilde{\ell}} Q \text{ then } \mathcal{A} \models Q \text{ and } \{\tilde{\ell}\} \text{ consistent with } \mathcal{A}$$

which expresses that the acceptability of analysis estimates is preserved by the reaction relation. Full semantic correctness, sometimes called *semantic soundness* follows directly by transitive closure:

$$\text{if } \mathcal{A} \models P \text{ and } P \xrightarrow{\tilde{L}}{}^{\star} Q \text{ then } \mathcal{A} \models Q \text{ and } \tilde{L} \text{ consistent with } \mathcal{A}$$

In order to formulate the required result we first expand $\mathcal{I}$ into the new relation $\mathcal{I}@\mathcal{R}$, which takes into account the bindings of variables specified by the $\mathcal{R}$ component:

$$\text{If } M^{\ell} \in \mathcal{I}(\mu), x \in \mathsf{fn}(M) \text{ and } n \in \langle\mathcal{R}\rangle(x) \text{ then } M^{\ell}[{}^{n}/_{x}] \in \mathcal{I}@\mathcal{R}(\mu).$$

It is immediate to show that $\mathcal{I}@\mathcal{R} = (\mathcal{I}@\mathcal{R})@\mathcal{R}$ and semantic correctness then specialises as follows:

$$
\begin{aligned}
\mathsf{prfx}_\Gamma((n)\,P) &= \mathsf{prfx}_\Gamma(P)\\
\mathsf{prfx}_\Gamma(\llbracket\, P \,\rrbracket^{\mu}) &= \mathsf{prfx}_\Gamma(P)\\
\mathsf{prfx}_\Gamma(P_1 \mid P_2) &= \mathsf{prfx}_\Gamma(P_1) \cup \mathsf{prfx}_\Gamma(P_2)\\
\mathsf{prfx}_\Gamma(\sum_{i\in I} M_i^{\ell_i}\,.\,P_i) &= \bigcup_{i\in I}(\{\ell_i\} \cup \mathsf{prfx}_\Gamma(P_i))\\
\mathsf{prfx}_\Gamma(\mathsf{rec}\,X.\,P) &= \mathsf{prfx}_{\Gamma[X\mapsto\emptyset]}(P)\\
\mathsf{prfx}_\Gamma(X) &= \Gamma(X)
\end{aligned}
$$

Fig. 16. Prefixes, $\mathsf{prfx}_\Gamma(P)$, of process P.

Theorem 4 (Semantic correctness). *Assume that $P_\star$ is a well-formed initial program, then $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\top} P_\star$ and $P_\star \xrightarrow{\tilde{L}}{}^{\star} P$ entails $(\mathscr{I}@\mathscr{R},\mathscr{R},\mathscr{F}) \models^{\top} P$ and $\tilde{L} \subseteq \mathscr{F}$.*

Proof. The corollary follows by induction on the length of $\tilde{L}$. The base case holds vacuously. The inductive step is established using Corollary 35 and Corollary 1 of Appendix B, the induction hypothesis, and the above insight. □

Intuitively, this asserts that an analysis estimate acceptable for $P_\star$ is acceptable for any process Q derivable by a sequence of reactions.

Implementability

Finally, it is desirable for every program, P, to actually have an acceptable analysis estimate and, indeed, a unique least such. This is the case if

$\{\mathcal{A} \mid \mathcal{A} \models P\}$ constitutes a Moore Family for all P

meaning that for all P

$\forall S' \subseteq \{\mathcal{A} \mid \mathcal{A} \models P\} : \sqcap S' \in \{\mathcal{A} \mid \mathcal{A} \models P\}$.

Trivially, a Moore family is never empty as it always contains a greatest element $\sqcap\emptyset = \top_{\mathfrak{A}}$, which is the trivial *worst* (i.e., least informative) acceptable analysis estimate. In contrast it is also guaranteed to have a least element $\mathsf{A} = \sqcap\{\mathcal{A} \mid \mathcal{A} \models P\}$, which is the least admissible result under the ordering $\sqsubseteq_{\mathfrak{A}}$ of $\mathfrak{A}$ and, hence, the *best* (most informative) acceptable estimate.

In the present case we do indeed have:

Theorem 5 (Moore family). *For any program $P_\star$ the set of acceptable analyses under $\models^{\mu}$ is a Moore family, i.e.*

$\forall S' \subseteq \{(\mathscr{I},\mathscr{R},\mathscr{F}) \mid (\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P_\star\} : \sqcap S' \in \{(\mathscr{I},\mathscr{R},\mathscr{F}) \mid (\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P_\star\}$.

Proof. The proof proceeds by structural induction in P. □

We shall write $(\mathscr{I}_\star,\mathscr{R}_\star,\mathscr{F}_\star) = \sqcap\{(\mathscr{I},\mathscr{R},\mathscr{F}) \mid (\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P_\star\}$ for the best estimate guaranteed by Theorem 5.

$$
\begin{aligned}
\mathsf{CP}_{\Gamma\Delta}((n)\,P) &= \mathsf{CP}_{\Gamma\Delta}(P)\\
\mathsf{CP}_{\Gamma\Delta}(\llbracket\, P \,\rrbracket^{\mu}) &= \mathsf{CP}_{\Gamma\Delta}(P)\\
\mathsf{CP}_{\Gamma\Delta}(P \mid P') &= \mathsf{CP}_{\Gamma\Delta}(P) \cup \mathsf{CP}_{\Gamma\Delta}(P') \cup \mathsf{cross}(P,P')\\
\mathsf{CP}_{\Gamma\Delta}(\sum_{i\in I} M_i^{\ell_i}\,.\,P_i) &= \bigcup_{i\in I}(\mathsf{CP}_{\Gamma\Delta}(P_i))\\
\mathsf{CP}_{\Gamma\Delta}(\mathsf{rec}\,X.\,P) &= \mathsf{CP}_{\Gamma[X\mapsto\mathsf{prfx}_{\Gamma[X\mapsto\emptyset]}(P)]\Delta[X\mapsto\emptyset]}(P)\\
\mathsf{CP}_{\Gamma\Delta}(X) &= \Delta(X)\\
\mathsf{cross}(P,P') &= \{(\ell_1,\ell_2) \mid (\ell_1,\ell_2) \in ((\mathsf{prfx}_\Gamma(P)\times\mathsf{prfx}_\Gamma(P')) \cup (\mathsf{prfx}_\Gamma(P')\times\mathsf{prfx}_\Gamma(P)))\}
\end{aligned}
$$

Fig. 17. *Concurrently Possible* capabilities, $\mathsf{CP}_{\Gamma\Delta}(P)$, of process P.

3.6. Implementation

The actual computation of $(\mathscr{I}_\star,\mathscr{R}_\star,\mathscr{F}_\star)$ is made possible by a simple change of viewpoint: Instead of using the acceptability judgement to check a given estimate $(\mathscr{I},\mathscr{R},\mathscr{F})$ against a process P we use it to derive a proof obligation from P. Algorithmically we capture this change of viewpoint as a clause generation function that takes as input a BioAmbients process, $P_\star$; it produces a formula, ϕ, over the symbols I, R, and F in our fragment of First Order Logic (FOL) such that $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\top} P_\star$ holds if, and only if, ϕ is true when interpreting I as $\mathscr{I}$, R as $\mathscr{R}$, and F as $\mathscr{F}$. In our case we can use the Alternation-free Least Fixed Point (ALFP) fragment [26] of FOL and this suffices for generating a polynomial time procedure for computing $(\mathscr{I}_\star,\mathscr{R}_\star,\mathscr{F}_\star)$. In practise we make use of a fixed point engine like our own 'The Succinct Solver Suite' [27,26].

4. Computing and preserving exposed prefixes

The 0CFA analysis of the previous section safely approximates the set of spatial configurations that may arise at run-time. In contrast it produces no information about the sequential order of the transitions that lead to the recorded configurations.

This lack of information strongly motivates the development of our main contribution: a flow sensitive *pathway analysis* that, given a program $P_\star$, computes a finite automaton that safely approximates the set of possibly infinite sequential behaviours that occur at run-time.

When developing the pathway analysis our focus shall be on the notion of *exposed prefixes*. Intuitively, the exposed prefixes of a process (or state) are those that might participate in the next reaction. Consider, for example, the system configuration

$$P = (n!\{m\}^{\ell_1} . m?\{q\}^{\ell_2} . P + \bullet?\{\bullet\}^{\ell_3} . Q) \,\big|\, n?\{p\}^{\ell_4} . m?\{q\}^{\ell_2} . R$$

where we write '$\bullet$' for an arbitrary name. Here there is one exposed occurrence of each of the prefixes $n!\{m\}^{\ell_1}$, $\bullet?\{\bullet\}^{\ell_3}$, and $n?\{p\}^{\ell_4}$. Due to their syntactic positions relative to one-another, the exposed prefixes, $n!\{m\}^{\ell_1}$ and $n?\{p\}^{\ell_4}$, enable a reaction, $P \xrightarrow{(\ell_1,\ell_4)} Q$, such that the resulting configuration,

$$Q = m?\{q\}^{\ell_2} . P \,\big|\, m?\{q\}^{\ell_2} . R,$$

only has two occurrences of $m?\{q\}^{\ell_2}$ as exposed prefixes.

Thus, for the purposes of the analysis, we shall abstractly characterise system configurations (or states) by their *extended multisets* of exposed prefixes. We then develop the desired automaton by tracking how these multisets evolve when reactions occur – a task that is complicated by the fact that a reaction may cause some exposed prefixes to disappear, and others to emerge.

Technically, we will turn to the *monotone frameworks*, normally associated with *Data Flow Analysis*, in order to deal with this. In particular we take inspiration from the classical *bit vector frameworks* (see e.g. [23]) where the flow of data from one *basic block* of program statements to the next is specified by transfer functions of the form:

$$f_{\mathsf{block}}(E) = (E \backslash kill_{\mathsf{block}}) \cup gen_{\mathsf{block}}$$

where, in a *forward analysis*, E is the information holding at the entry to the block, $kill_{\mathsf{block}}$ is the information invalidated by the block, and gen_{block} is the information created by the block.

For the purpose of the pathway analysis we shall attach transfer functions to the reactions of processes. Corresponding to a forward data flow analysis, we then rely on these transfer functions to compute how the extended multisets of exposed prefixes are transformed as reactions occur. So, much akin to bit vector frameworks our transfer functions will take the simple form

$$f_{\mathsf{react}}(E) = (E \backslash kill_{\mathsf{react}}) \cup gen_{\mathsf{react}}$$

where E is the extended multiset of prefixes exposed before the reaction, $kill_{\mathsf{react}}$ is the extended multiset of prefixes discarded by the reaction, and gen_{react} is the extended multiset of prefixes exposed by the reaction. But, in contrast to the simple powersets associated with bit vector frameworks, the extended multisets are able to keep track of the *number* of exposed prefixes.

In the following we start by introducing the notion of extended multisets in Section 4.1. In Section 4.2 we then go on to specify the extended multiset of exposed prefixes, $\mathcal{E}_\star[\![P]\!]$, of a given process, P. In Section 4.3 we specify the multisets of prefixes, $\mathcal{G}_\star[\![P]\!]$, generated by the various reactions within a given process, P. Similarly, we specify the multisets, $\mathcal{K}_\star[\![P]\!]$, killed by the reactions of a given process, P, in Section 4.4. Finally, in Section 4.5, we specify a transfer function, $\mathsf{transfer}_{P,\tilde{\ell}}(E)$, to capture the effect of each reaction of P.

Note that, in contrast to the Flow Logic approach, the approach of monotone frameworks offers no convenient separation between specification and implementation. Thus, once we have defined the appropriate transfer functions, we shall turn to the issue of implementation and define a suitable worklist algorithm in Section 5.

4.1. Extended multisets

We define an *extended multiset*, M, as an element of the domain

$$\mathfrak{M} = \mathbf{Lab} \to (\mathbb{N} \cup \infty)$$

which, intuitively, allows each primitive prefix of a process (identified by its unique label) to be associated with a value corresponding to a number of occurrences (or ∞ when the number might be unbounded). The domain is equipped with an ordering $\leq_{\mathfrak{M}}$ defined by

$$\mathsf{M} \leq_{\mathfrak{M}} \mathsf{M}' \quad \text{iff} \quad \forall \ell : \mathsf{M}(\ell) \leq \mathsf{M}'(\ell) \vee \mathsf{M}'(\ell) = \infty$$

The domain $(\mathfrak{M}, \leq_{\mathfrak{M}})$ constitutes a complete lattice where the constants and operations corresponding to $\bot, \top, \sqcap, \sqcup$ are defined as follows:

$$\bot_{\mathfrak{M}} \text{ is defined by } \quad \forall \ell : \bot_{\mathfrak{M}}(\ell) = 0$$

$$\top_{\mathfrak{M}} \text{ is defined by } \quad \forall \ell : \top_{\mathfrak{M}}(\ell) = \infty$$

$$\min_{\mathfrak{M}} \text{ is defined by } \quad (\mathsf{M} \min_{\mathfrak{M}} \mathsf{M}')(\ell) = \begin{cases} \min\{\mathsf{M}(\ell), \mathsf{M}'(\ell)\} & \text{if } \mathsf{M}(\ell) \in \mathbb{N} \wedge \mathsf{M}'(\ell) \in \mathbb{N} \\ \mathsf{M}(\ell) & \text{if } \mathsf{M}'(\ell) = \infty \\ \mathsf{M}'(\ell) & \text{otherwise} \end{cases}$$

$$\max_{\mathfrak{M}} \text{ is defined by } (\mathsf{M} \max_{\mathfrak{M}} \mathsf{M}')(\ell) = \begin{cases} \max\{\mathsf{M}(\ell), \mathsf{M}'(\ell)\} & \text{if } \mathsf{M}(\ell) \in \mathbb{N} \wedge \mathsf{M}'(\ell) \in \mathbb{N} \\ \infty & \text{otherwise} \end{cases}$$

Furthermore we define addition and subtraction on multisets as follows:

$$+_{\mathfrak{M}} \text{ is defined by } (\mathsf{M} +_{\mathfrak{M}} \mathsf{M}')(\ell) = \begin{cases} \mathsf{M}(\ell) + \mathsf{M}'(\ell) & \text{if } \mathsf{M}(\ell) \in \mathbb{N} \wedge \mathsf{M}'(\ell) \in \mathbb{N} \\ \infty & \text{otherwise} \end{cases}$$

$$-_{\mathfrak{M}} \text{ is defined by } (\mathsf{M} -_{\mathfrak{M}} \mathsf{M}')(\ell) = \begin{cases} \mathsf{M}(\ell) - \mathsf{M}'(\ell) & \text{if } \mathsf{M}(\ell) \in \mathbb{N} \wedge \mathsf{M}(\ell) \geq \mathsf{M}'(\ell) \\ 0 & \text{if } \mathsf{M}(\ell) \in \mathbb{N} \wedge \mathsf{M}(\ell) < \mathsf{M}'(\ell) \\ 0 & \text{if } \mathsf{M}(\ell) \in \mathbb{N} \wedge \mathsf{M}'(\ell) = \infty \\ \infty & \text{otherwise} \end{cases}$$

In particular note that $\infty - \infty$ is intended to give ∞ in order to ensure that the transfer function defined in Section 4.5 is always a safe over-approximation.

4.2. Exposed prefixes

Using the extended multisets we shall now define the concept of *exposed prefixes*. We formalise this as a function

$$\mathcal{E}_{\bigstar}[\![\ ]\!] : \mathbf{Proc} \to \mathfrak{M}$$

intended to capture the following intuition: For a process P the expression $\mathcal{E}_{\bigstar}[\![P]\!](\ell)$ denotes the number of distinct occurrences of ℓ that might participate in the next reaction $P \xrightarrow{\tilde{\ell}} Q$.

Recall that our focus is not on a given process P but on all processes congruent to P; in particular those that arise by the unfolding of recursion. In order to appropriately address this issue we use the Γ indexed function $\mathcal{E}_{\Gamma}[\![\ ]\!]$ shown in Fig. 18 when computing the exposed prefixes. The index $\Gamma : \mathbf{Pid} \to \mathfrak{M}$ is a mapping that associates an extended multiset with each free process identifier. It is needed in order to support the least fixed point computation that is required for computing the exposed prefixes of recursive processes; it simply unfolds the recursion until no further exposed prefixes arise from doing so.

It is obvious, e.g. if $P = \mathrm{rec}X.M^{\ell}.P' \,|\, X$, that the naive computation may not terminate because the unfolding process could go on indefinitely and the domain, $(\mathfrak{M}, \leq_{\mathfrak{M}})$, admits infinite ascending chains. However, it turns out that we are justified in recasting the computation in terms of the *expansion operator* $\bowtie_{\mathfrak{M}}$, also shown in Fig.18, which ensures termination in two iterations. We refer to [31] for a formal proof of this result but provide an informal argument below.

$$
\begin{aligned}
\mathcal{E}_\Gamma[\![\,(n)\,P\,]\!] &= \mathcal{E}_\Gamma[\![\,P\,]\!] \\
\mathcal{E}_\Gamma[\![\,[\,P\,]^{\mu}\,]\!] &= \mathcal{E}_\Gamma[\![\,P\,]\!] \\
\mathcal{E}_\Gamma[\![\,P_1 \,|\, P_2\,]\!] &= \mathcal{E}_\Gamma[\![\,P_1\,]\!] +_{\mathfrak{M}} \mathcal{E}_\Gamma[\![\,P_2\,]\!] \\
\mathcal{E}_\Gamma[\![\sum_{i\in I} M_i^{\ell_i}\,.\,P_i\,]\!] &= +_{\mathfrak{M}\,i\in I} \bot_{\mathfrak{M}}[\ell_i \mapsto 1] \\
\mathcal{E}_\Gamma[\![\,\mathsf{rec}\,X.\,P\,]\!] &= \mathrm{LFP}(\lambda E.\ \mathcal{E}_{\Gamma[X\mapsto E]}[\![\,P\,]\!]) \\
&= \mathcal{E}_{\Gamma[X\mapsto \bot_{\mathfrak{M}}]}[\![\,P\,]\!] \bowtie_{\mathfrak{M}} \mathcal{E}_{\Gamma[X\mapsto \mathcal{E}_{\Gamma[X\mapsto \bot_{\mathfrak{M}}]}[\![\,P\,]\!]]}[\![\,P\,]\!] \\
\mathcal{E}_\Gamma[\![\,X\,]\!] &= \Gamma(X)
\end{aligned}
$$

$$
(M \bowtie_{\mathfrak{M}} M')(\ell) = \begin{cases} M(\ell) & \text{if } M(\ell) = M'(\ell) \\ \infty & \text{otherwise} \end{cases}
$$

Fig. 18. Definition of $\mathcal{E}_\Gamma[\![\,P\,]\!]$.

Informally it is easy to see that *if* a process identifier X occurs un-guarded in the body of $\mathsf{rec}\,X.P$, e.g. in $\mathsf{rec}\,X.(X \,|\, P)$, then the exposed prefixes of P will have infinitely many occurrences and the naive computation of the fixed point never terminates. However, it is also the case that the number of exposed prefixes will grow in every iteration *only if* the X is indeed unguarded. Obviously, two iterations suffice in order to determine this kind of behaviour.

Apart from this technicality the function $\mathcal{E}_\Gamma[\![\ ]\!]$ performs a rather straightforward recursive descent into processes using the addition operation of extended multisets in order to calculate the total number of exposed reaction prefixes, i.e. those that could engage in reaction. As should be evident from the case concerning guarded sums only the top-most prefixes contribute and the result is obtained by addition of their multiplicities.

Finally, the desired function $\mathcal{E}_\bigstar[\![P]\!]$ is defined by

$$\mathcal{E}_\bigstar[\![P]\!] = \mathcal{E}_{[\,]}[\![\,P\,]\!]$$

When P is an initial program $P_\bigstar$ we shall use the distinguished symbol $E_\bigstar$ to denote $\mathcal{E}_\bigstar[\![P]\!]$.

Example 4. Consider Example 1. Here the blood Filter in line 1 exposes expel $rj^1 \times 1$, the food Transport mechanism in line 2 exposes $\{rea\hat{}?\{rl\}^2 \times 1, \mathsf{exit}\ rj^4 \times 1, \mathsf{enter}\ ac^5 \times 1\}$ whereas expel rl^3 is hidden behind $rea\hat{}?\{rl\}^2$, the Nutrient in line 3 exposes exit $RL^6 \times 1$, and the Cell exposes $\{rea\_!\{RL\}^7 \times 1, \mathsf{expel}\ rj^8 \times 1, \mathsf{accept}\ ac^9 \times 1\}$. Hence, the result, E_{eat}, of subjecting the program P_{eat} to the exposed capability analysis basically emerges as the multiset sum of these exposed occurrences and is shown below:

$$E_{\mathsf{eat}} = \bot_{\mathfrak{M}}[1 \mapsto 1, 2 \mapsto 1, 4 \mapsto 1, 5 \mapsto 1, 6 \mapsto 1, 7 \mapsto 1, 8 \mapsto 1, 9 \mapsto 1]$$

One may observe that exactly one copy of every capability except for 3 is exposed.

Intuitively, *correctness of* $\mathcal{E}_\Gamma[\![\,P\,]\!]$ means (1) that it is invariant under the heating relation and (2) that it correctly captures the prefixes that may be involved in the first reaction step:

Lemma 6. *If* $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $P \Rightarrow Q$ *then* $\mathcal{E}_\Gamma[\![\,Q\,]\!] = \mathcal{E}_\Gamma[\![\,P\,]\!]$ *and furthermore, if* $P \xrightarrow{(\ell_1,\ell_2)} Q$ *then* $\ell_1 \in \mathit{dom}(\mathcal{E}_\Gamma[\![\,P\,]\!])$ *and* $\ell_2 \in \mathit{dom}(\mathcal{E}_\Gamma[\![\,P\,]\!])$.

Proof. The result emerges as a corollary of Lemma 38 in Appendix C. □

4.3. Generated prefixes

In preparation for the transfer function we shall define a notion of generate functions

$$\mathcal{G}_\bigstar[\![\ ]\!] : \mathbf{Proc} \to \mathfrak{T}$$

where elements of type

$$\mathfrak{T} = (\mathbf{Lab} \to \mathfrak{M})$$

$$
\begin{aligned}
\mathcal{G}_{\Gamma\Delta}[\![\,(n)\,P\,]\!] &= \mathcal{G}_{\Gamma\Delta}[\![\,P\,]\!] \\
\mathcal{G}_{\Gamma\Delta}[\![\,\lfloor\, P \,\rfloor^{\mu}\,]\!] &= \mathcal{G}_{\Gamma\Delta}[\![\,P\,]\!] \\
\mathcal{G}_{\Gamma\Delta}[\![\,P_1 \mid P_2\,]\!] &= \mathcal{G}_{\Gamma\Delta}[\![\,P_1\,]\!] \;\mathsf{max}_{\mathfrak{T}}\; \mathcal{G}_{\Gamma\Delta}[\![\,P_2\,]\!] \\
\mathcal{G}_{\Gamma\Delta}[\![\sum_{i\in I} M_i^{\ell_i}.\,P_i\,]\!] &= \mathsf{MAX}_{\mathfrak{T}\,i\in I}(\bot_{\mathfrak{T}}[\ell_i \mapsto \mathcal{E}_{\Gamma}[\![\,P_i\,]\!]] \;\mathsf{max}_{\mathfrak{T}}\; \mathcal{G}_{\Gamma\Delta}[\![\,P_i\,]\!]) \\
\mathcal{G}_{\Gamma\Delta}[\![\,\mathsf{rec}\,X.\,P\,]\!] &= \mathrm{LFP}(\lambda G.\; \mathcal{G}_{\Gamma[X\mapsto\mathcal{E}_{\Gamma}[\![\,\mathsf{rec}\,X.\,P\,]\!]\Delta[X\mapsto G]}[\![\,P\,]\!]) \\
&= \mathcal{G}_{\Gamma[X\mapsto\mathcal{E}_{\Gamma}[\![\,\mathsf{rec}\,X.\,P\,]\!]\Delta[X\mapsto\bot_{\mathfrak{T}}]}[\![\,P\,]\!] \\
\mathcal{G}_{\Gamma\Delta}[\![\,X\,]\!] &= \Delta(X)
\end{aligned}
$$

Fig. 19. Definition of $\mathcal{G}_{\Gamma\Delta}[\![\,P\,]\!]$.

are mappings from labels into extended multisets. The intuition intended is the following: For a process P such that $P \xrightarrow{(\ell_1,\ell_2)} Q$ the expression $\mathcal{G}_{\bigstar}[\![P]\!]\,(\ell_1)(\ell)$ denotes a safe (over-) approximation to the number of occurrences of ℓ that may become exposed in Q due to the involvement of ℓ_1 in the transition from P. Note, that we define $\mathcal{G}_{\bigstar}[\![P]\!]\,((\ell_1,\ell_2)) = \mathcal{G}_{\bigstar}[\![P]\!]\,(\ell_1) +_{\mathfrak{M}} \mathcal{G}_{\bigstar}[\![P]\!]\,(\ell_2)$.

The domain $\mathfrak{T}$ is a straightforward extension of $\mathfrak{M}$ and the constants $\bot_{\mathfrak{T}}$,$\top_{\mathfrak{T}}$ are defined as expected. The associated operations $\leq_{\mathfrak{T}}$, $\mathsf{min}_{\mathfrak{T}}$, $\mathsf{max}_{\mathfrak{T}}$,$+_{\mathfrak{T}}$, and $-_{\mathfrak{T}}$ are all defined as point-wise extensions of the corresponding operators on $\mathfrak{M}$, e.g. $\leq_{\mathfrak{T}}$ is defined by

$$T_1 \leq_{\mathfrak{T}} T_2 \quad \text{iff } \forall \ell : T_1(\ell) \leq_{\mathfrak{M}} T_2(\ell)$$

Once more we have to take the unfolding of recursion into account when computing the generate function. For this we use the Γ and Δ indexed recursive procedure $\mathcal{G}_{\Gamma\Delta}[\![\ ]\!]$ defined in Fig. 19. Here Γ : **Pid** $\to \mathfrak{M}$ is as in $\mathcal{E}_{\Gamma}[\![\ ]\!]$, i.e. a mapping that associates an extended multiset with each free process identifier, and Δ : **Pid** $\to \mathfrak{T}$ is a mapping that associates a mapping from labels into extended multisets with each free process identifier. We use the Γ index to ensure that the number of prefixes exposed in a given continuation is computed correctly, i.e. in the case of $\mathsf{rec}\,X.\,P$, Γ is used to appropriately associate X with the multiset of prefixes $\mathcal{E}_{\Gamma}[\![\,\mathsf{rec}\,X.\,P\,]\!]$ exposed by the recursion construct. The Δ index, on the other hand, is used to support the fixed point computation required for $\mathsf{rec}\,X.\,P$ – much like the Γ index in the definition of $\mathcal{E}_{\bigstar}[\![\ ]\!]$.

The interesting case is that of the guarded sum constructs $\sum_{i\in I} M_i^{\ell_i}.\,P_i$. It is straightforward to see that every guard $M_i^{\ell_i}$ generates all of the prefixes exposed by the corresponding continuation P_i. However, each distinct ℓ may have several occurrences in a process P and, as we are aiming for a safe over-approximation, we must take this into account. We do so by ensuring that the results obtained for sub-expressions are combined using $\mathsf{max}_{\mathfrak{T}}$. This ensures that the computed $\mathcal{G}_{\Gamma\Delta}[\![\,P\,]\!]$ is a safe over-approximation - in the sense that for every ℓ, $\mathcal{G}_{\Gamma\Delta}[\![\,P\,]\!](\ell)$ safely over-approximates every multiset generated by a specific occurrence of ℓ in P – even if labels are not unique.

In the case of the recursion construct, however, we have to unfold the recursion until no further information about generated prefixes arises from doing so. This amounts to the least fixed point computation shown in Fig. 19. Again, the termination of the naive computation is endangered because $(\mathfrak{T},\leq_{\mathfrak{T}})$ admits infinite ascending chains. As formally proved in [31], however, the computation actually stabilises after a single iteration, which justifies the alternative formulation of Fig. 19.

The desired function $\mathcal{G}_{\bigstar}[\![P]\!]$ is defined by

$$\mathcal{G}_{\bigstar}[\![P]\!] = \mathcal{G}_{[\,][\,]}[\![\,P\,]\!]$$

When P is an initial program $P_{\bigstar}$ we shall use the distinguished symbol $G_{\bigstar}$ to denote $\mathcal{G}_{\bigstar}[\![P]\!]$.

Example 5. The result G_{eat} of subjecting the program P_{eat} to the generated capability analysis is shown below:

$$
\begin{aligned}
G_{\mathsf{eat}} = \bot_{\mathfrak{T}}[\quad & 1 \mapsto \bot_{\mathfrak{M}}[1 \mapsto 1], \\
& 2 \mapsto \bot_{\mathfrak{M}}[3 \mapsto 1], \\
& 3 \mapsto \bot_{\mathfrak{M}}[2 \mapsto 1, 4 \mapsto 1, 5 \mapsto 1] \\
& 4 \mapsto \bot_{\mathfrak{M}}[2 \mapsto 1, 4 \mapsto 1, 5 \mapsto 1] \\
& 5 \mapsto \bot_{\mathfrak{M}}[2 \mapsto 1, 4 \mapsto 1, 5 \mapsto 1] \\
& 7 \mapsto \bot_{\mathfrak{M}}[7 \mapsto 1, 8 \mapsto 1, 9 \mapsto 1] \\
& 8 \mapsto \bot_{\mathfrak{M}}[7 \mapsto 1, 8 \mapsto 1, 9 \mapsto 1] \\
& 9 \mapsto \bot_{\mathfrak{M}}[7 \mapsto 1, 8 \mapsto 1, 9 \mapsto 1]]
\end{aligned}
$$

Labels mapped to $\bot_{\mathfrak{M}}$ are left out in the enumeration of the multiset.

$$
\begin{aligned}
\mathcal{K}_\Delta[\![\,(n)\,P\,]\!]env &= \mathcal{K}_\Delta[\![\,P\,]\!] \\
\mathcal{K}_\Delta[\![\,\lfloor P \rfloor^{\mu}\,]\!] &= \mathcal{K}_\Delta[\![\,P\,]\!] \\
\mathcal{K}_\Delta[\![\,P_1 \mid P_2\,]\!] &= \mathcal{K}_\Delta[\![\,P_1\,]\!] \min_{\mathfrak{T}} \mathcal{K}_\Delta[\![\,P_2\,]\!] \\
\mathcal{K}_\Delta[\![\,\sum_{i\in I} M_i^{\ell_i}\,.\,P_i\,]\!] &= \mathsf{MIN}_{\mathfrak{T}\,i\in I}(\top_{\mathfrak{T}}[\ell_i \mapsto M] \min_{\mathfrak{T}} \mathcal{K}_\Delta[\![\,P_i\,]\!]) \\
&\qquad \text{where } M = +_{\mathfrak{M}\,j\in I}(\bot_{\mathfrak{M}}[\ell_j \mapsto 1]) \\
\mathcal{K}_\Delta[\![\,\mathsf{rec}\,X.\,P\,]\!] &= \mathrm{LFP}(\lambda K.\mathcal{K}_{\Delta[X\mapsto K]}[\![\,P\,]\!]) \\
\mathcal{K}_\Delta[\![\,X\,]\!] &= \Delta(X)
\end{aligned}
$$

Fig. 20. Definition of kill functions.

Due to the different role that $\mathcal{G}_{\Gamma\Delta}[\![\ ]\!]$ plays in the transfer function the *correctness of* $\mathcal{G}_{\Gamma\Delta}[\![\,P\,]\!]$ is slightly more involved than was the case for $\mathcal{E}_\Gamma[\![\ ]\!]$. Surely, $\mathcal{G}_{\Gamma\Delta}[\![\ ]\!]$ must be invariant under heating.

Lemma 7. *If* $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $P \Rrightarrow Q$ *then* $\mathcal{G}_{\Gamma\Delta}[\![\,Q\,]\!] = \mathcal{G}_{\Gamma\Delta}[\![\,P\,]\!]$.

Proof. The lemma is a corollary of Lemma 40 in Appendix C. □

Finally we must show that the safety of the approximation is preserved under reduction. This amounts to the following 'local subject reduction result:

Lemma 8. *If* $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $P \xrightarrow{\tilde{\ell}} Q$ *then* $\mathcal{G}_{\Gamma\Delta}[\![\,Q\,]\!] \leq_{\mathfrak{T}} \mathcal{G}_{\Gamma\Delta}[\![\,P\,]\!]$.

Proof. The result emerges as a corollary of Theorem 41 in Appendix C. □

4.4. Killed prefixes

As the last component of the transfer function we shall also define a notion of kill functions

$$\mathcal{K}_\star[\![\ ]\!] : \mathbf{Proc} \to \mathfrak{T}$$

the intention of which is the following: For a process P such that $P \xrightarrow{(\ell_1,\ell_2)} Q$ the expression $\mathcal{K}_\star[\![P]\!](\ell_1)(\ell)$ denotes a safe (under-) approximation to the number of occurrences of ℓ that are no longer exposed in Q because of the involvement of ℓ_1 in the transition from P. Note again, that we define $\mathcal{K}_\star[\![P]\!]((\ell_1,\ell_2)) = \mathcal{K}_\star[\![P]\!](\ell_1) + \mathcal{K}_\star[\![P]\!](\ell_2)$.

The unfolding of recursion also complicates the computation of kill information and therefore we use the Δ indexed recursive procedure $\mathcal{K}_\Delta[\![\ ]\!]$ defined in Fig. 20 for the computation. Unlike $\mathcal{G}_{\Gamma\Delta}[\![\ ]\!]$ we do not need the Γ index (as in $\mathcal{E}_\Gamma[\![\ ]\!]$) because knowledge of exposed actions is not needed for defining $\mathcal{K}_\Delta[\![\ ]\!]$. As before $\Delta : \mathbf{Pid} \to \mathfrak{T}$ serves as a mapping that associates a mapping from labels into extended multisets with each free process identifier. Similar to before it is used to support the fixed point computation required for $\mathsf{rec}\,X.\,P$.

Again, the interesting case is that of guarded sum constructs $\sum_{i\in I} M_i^{\ell_i}\,.\,P_i$. It is straightforward to see that every guard of such a construct kills exactly one occurrence of each guard in the construct, including itself. It is still the case, however, that each distinct ℓ may have several occurrences in a process P. We ensure the safety of the computed under-approximation by combining the results obtained for sub-expressions using $\min_{\mathfrak{T}}$. As a consequence the computed $\mathcal{K}_\Delta[\![\ ]\!]$ is always a safe approximation – in the sense that for every ℓ, $\mathcal{K}_\Delta[\![\,P\,]\!](\ell)$ safely under-approximates every multiset killed by a specific occurrence of ℓ in P – even when labels are not unique.

As usual the recursion construct requires a fixed point computation where unfolding is performed until no further information about killed prefixes arises from doing so. This amounts to the least fixed point computation shown in Fig. 20, which is guaranteed to terminate because $\mathfrak{T}$ admits no infinite descending chains. In other words, the multiset of killed prefixes is *shrunk* by the suggested iterative procedure, hence the iteration surely stops when the multiset is empty.

Again, the desired function $\mathcal{K}_\star[\![P]\!]$ is defined by

$$\mathcal{K}_\star[\![P]\!] = \mathcal{K}_{[\,]}[\![\,P\,]\!]$$

When P is an initial program $P_\star$ we shall use the distinguished symbol $K_\star$ to denote $\mathcal{K}_\star[\![P]\!]$.

Example 6. The result K_{eat} of subjecting the program P_{eat} to the killed capability analysis is shown below:

$$\begin{array}{rl} K_{\mathsf{eat}} = \top_{\mathfrak{T}}[& 1 \mapsto \bot_{\mathfrak{M}}[1 \mapsto 1] \\ & 2 \mapsto \bot_{\mathfrak{M}}[2 \mapsto 1\ 4 \mapsto 1\ 5 \mapsto 1], \\ & 3 \mapsto \bot_{\mathfrak{M}}[3 \mapsto 1] \\ & 4 \mapsto \bot_{\mathfrak{M}}[2 \mapsto 1\ 4 \mapsto 1\ 5 \mapsto 1], \\ & 5 \mapsto \bot_{\mathfrak{M}}[2 \mapsto 1\ 4 \mapsto 1\ 5 \mapsto 1], \\ & 6 \mapsto \bot_{\mathfrak{M}}[6 \mapsto 1] \\ & 7 \mapsto \bot_{\mathfrak{M}}[7 \mapsto 1\ 8 \mapsto 1\ 9 \mapsto 1], \\ & 8 \mapsto \bot_{\mathfrak{M}}[7 \mapsto 1\ 8 \mapsto 1\ 9 \mapsto 1], \\ & 9 \mapsto \bot_{\mathfrak{M}}[7 \mapsto 1\ 8 \mapsto 1\ 9 \mapsto 1]] \end{array}$$

Besides of being an under- rather than an over-approximation the *correctness of* $\mathcal{K}_\Delta[\![\ ]\!]$ is rather similar to that of $\mathcal{G}_{\Gamma\Delta}[\![\ ]\!]$. I.e. first we must ensure invariance under heating.

Lemma 9. *If* $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $P \Rrightarrow Q$ *then* $\mathcal{K}_\Delta[\![P]\!] = \mathcal{K}_\Delta[\![Q]\!]$.

Proof. The result is a corollary of Lemma 43 in Appendix C. □

Also, we must show that the safety of the approximation is preserved under reduction:

Lemma 10. *If* $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $P \xrightarrow{\tilde{\ell}} Q$ *then* $\mathcal{K}_\Delta[\![P]\!] \leq_{\mathfrak{T}} \mathcal{K}_\Delta[\![Q]\!]$.

Proof. The result is a corollary of Lemma 44 in Appendix C. □

4.5. The transfer function

We follow the template of bit vector frameworks when defining the transfer functions of our setup. Here transitions serve the role of basic blocks so that E is the extended multiset of exposed prefixes characterising some state P where $P \xrightarrow{\tilde{\ell}} Q$ might be enabled for some Q, $\mathcal{K}_\bigstar[\![P]\!](\tilde{\ell})$ is the extended multiset of prefixes that is guaranteed to be disabled by the transition, and $\mathcal{G}_\bigstar[\![P]\!](\tilde{\ell})$ is the extended multiset of prefixes that might be enabled by the transition. Thus the transfer function takes the form:

$$\mathsf{transfer}_{P,\tilde{\ell}}(E) = (E -_{\mathfrak{M}} \mathcal{K}_\bigstar[\![P]\!](\tilde{\ell})) +_{\mathfrak{M}} \mathcal{G}_\bigstar[\![P]\!](\tilde{\ell})$$

Given a multiset, E, of exposed prefixes the transfer function computes, for an a priori given process P and transition $\tilde{\ell}$, a safe over-approximation of the set of prefixes exposed after the transition. Note that the computation is based on union and intersection of multisets rather than the ordinary sets that are usually used for bit vector frameworks. Also note that, since $\tilde{\ell}$ is a *pair* (ℓ_1,ℓ_2), $K(\tilde{\ell})$ actually denotes $(K(\ell_1)) +_{\mathfrak{M}} (K(\ell_2))$ and similarly for $G(\tilde{\ell})$.

The following result states that this transfer function provides a safe approximation to the exposed actions of the process that results from the transition:

Theorem 11 (Subject reduction)
If $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $P \xrightarrow{\tilde{\ell}} Q$ *then* $\mathcal{E}_\bigstar[\![Q]\!] \leq_{\mathfrak{M}} \mathsf{transfer}_{P,\tilde{\ell}}(\mathcal{E}_\bigstar[\![P]\!])$.

Proof. The theorem follows from Theorem 45 in Appendix C. □

Finally, we establish the semantic soundness of the transfer functions as a straightforward corollary of the theorem.

Corollary 12. *If* $\mathsf{PRG}_{\mathbf{C}}(P_\bigstar)$ *and* $P_\bigstar \xrightarrow{\tilde{L}}{}^{\bigstar} P \xrightarrow{\tilde{\ell}} Q$ *then* $\mathcal{E}_\bigstar[\![Q]\!] \leq_{\mathfrak{M}} \mathsf{transfer}_{P_\bigstar,\tilde{\ell}}(\mathcal{E}_\bigstar[\![P]\!])$.

Proof. This follows from Theorem 47 in Appendix C. □

This shows that the approximation is safe for all reaction sequences that may arise from an initial program $P_\bigstar$.

```
(1)  Q := {q⋆}; E[q⋆] := E⋆;
(2)  W := {q⋆}; δ := ∅;
(3)  while W ≠ ∅ do
(4)    select q_s from W; W := W\{q_s};
(5)    for each ℓ̃ ∈ enabled(E[q_s]) do
(6)      let E = transfer_{P⋆,ℓ̃}(E[q_s])
(7)      in update(q_s, ℓ̃, E)
```

Fig. 21. The worklist algorithm.

5. Constructing the automaton

We now turn to the pragmatics of calculating the finite automata that are the goals of the pathway analysis. Given a program $P_\bigstar$ the idea is to construct a finite automaton such that the potentially infinite transition system of $P_\bigstar$ is faithfully represented within the states and transitions of the automaton. The computed automaton will have the following components:

- A set, Q, of states. Each state, q, is associated with an extended multiset, $\mathsf{E}[q]$, and is intended to represent all processes, P, for which $\mathcal{E}_\bigstar[\![P]\!] \preceq_{\mathfrak{M}} \mathsf{E}[q]$.
- An initial state, $q_\bigstar \in \mathsf{Q}$, associated with the corresponding set, $E_\bigstar$, of exposed prefixes.
- A transition relation, δ, containing transitions, $q_s \overset{(\ell_1,\ell_2)}{\Longrightarrow} q_t$, reflecting that in the state q_s two processes, exposing prefixes labelled ℓ_1 and ℓ_2 respectively, may react and give rise to q_t.

We shall refer to the automaton as $(\mathsf{Q},q_\bigstar,\delta,\mathsf{E})$.

It will emerge from the construction that the resulting automaton is *partially deterministic* in the sense that if $q_s \overset{\tilde{\ell}}{\Longrightarrow} q_1$ and $q_s \overset{\tilde{\ell}}{\Longrightarrow} q_2$ then $q_1 = q_2$. This suffices for our purposes and hence we shall not make the effort of adding a fail state in order to obtain a deterministic finite automaton.

5.1. The worklist algorithm

Motivated by monotone frameworks [23] the heart of the computation is an iterative worklist algorithm that computes a solution to the framework instance given as input. It simply starts out from the initial state and constructs the automaton by adding more and more states and transitions.

The algorithm, which is defined in Fig. 21, computes over four data structures:

- A set Q of states.
- A vector E of associated multisets of exposed capabilities.
- A worklist $\mathsf{W} \subseteq \mathsf{Q}$ of states that have yet to be processed.
- A set δ of transitions valid for the current automaton.

The algorithm is initialised in line (1) and (2). First the start state, $q_\bigstar$, and associated multiset, $E_\bigstar$, of prefixes exposed by $P_\bigstar$ are added to the otherwise empty set of states, Q, and vector, E. Then the start state, $q_\bigstar$, is added to the worklist and the current set of transitions δ is set to the empty set.

Line (3) defines the iterative loop inspecting the worklist until it is finally empty. In every iteration, line (4) selects and removes a state, q_s, from the worklist. The set of reactions potentially causing transitions out of q_s is then computed using the procedure call, $\mathsf{enabled}(\mathsf{E}[q_s])$, in line (5). The corresponding procedure is defined in Section 5.2 and will yield a set of *pairs*, i.e. $\mathsf{enabled}(\mathsf{E}[q_s]) \subseteq \mathsf{E}[q_s] \times \mathsf{E}[q_s]$. For each potential reaction, $\tilde{\ell}$, an extended multiset, E, denoting the corresponding next state is computed in line (6) by a call, $\mathsf{transfer}_{P_\bigstar,\tilde{\ell}}(\mathsf{E}[q_s])$, using the transfer function of the previous section.

Finally, in line (7), the automaton is updated to reflect the new transition using a call $\mathsf{update}(q_s,\tilde{\ell},E)$ to the update procedure defined in Section 5.3. As we shall see it is crucial for termination that this update is done in a way such that Q remains finite. For this we shall enable extensive reuse of states using by a clever way of comparing them, and only if we fail in finding a suitable preexisting state do we add a new one.

5.2. Enabled reactions

Writing $E = \mathsf{E}[q]$ for the extended multiset associated with some state q we use the procedure $\mathsf{enabled}(E)$ to compute the set of potential reactions of q. If $\ell_1 \in \mathsf{dom}(E)$ and $\ell_2 \in \mathsf{dom}(E)$ then the pair (ℓ_1,ℓ_2) may be *enabled* in q if the corresponding prefixes

(i) *match* in the sense that one is complementary to the other, and
(ii) may be *concurrently possible*.

Inspecting the definition of $\mathscr{F}_\bigstar$ in the CFA of Section 3 (Figs. 14 and 15), it becomes clear that $\mathscr{F}_\bigstar$ precisely captures the intuitions suggesting (i) and (ii). This leads to estimating the set of enabled transitions simply by taking the pairs in $\mathscr{F}_\bigstar$

(1) if some $q \in \mathsf{Q}$ with $\mathsf{dom}(\mathsf{E}[q]) = \mathsf{dom}(E)$
(2) then $q_t := q$
(3) else select q_t from outside Q; $\mathsf{Q} := \mathsf{Q} \cup \{q_t\}$; $\mathsf{E}[q_t] := \bot_{\mathfrak{M}}$;
(4) if $\neg(E \leq_{\mathfrak{M}} \mathsf{E}[q_t])$
(5) then $\mathsf{E}[q_t] := \mathsf{E}[q_t] \nabla_{\mathfrak{M}} E$; $\mathsf{W} := \mathsf{W} \cup \{q_t\}$
(6) $\delta := (\delta \backslash \{(q_s, \tilde{\ell}, q) \mid q \in \mathsf{Q}\}) \cup \{(q_s, \tilde{\ell}, q_t)\}$;
(7) $\mathsf{clean\text{-}up}(\mathsf{Q}, \mathsf{W}, \delta)$

Fig. 22. $\mathsf{update}(q_s,\tilde{\ell},E)$.

where both of the two capabilities are exposed in the present state:

$$\mathsf{enabled}(E) = \{(\ell_1,\ell_2) \mid (\ell_1,\ell_2) \in \mathscr{F}_\star \wedge \ell_1 \in \mathsf{dom}(E) \wedge \ell_2 \in \mathsf{dom}(E)\}$$

The function enabled is correct in the sense that it safely over-approximates the set of enabled transitions:

Lemma 13. *If* $P_\star \xrightarrow{\tilde{L}}{}^\star P \xrightarrow{\tilde{\ell}} Q$ *and* $\mathcal{E}_\star[\![P]\!] \leq_{\mathfrak{M}} E$ *then* $\tilde{\ell} \in \mathsf{enabled}(E)$.

Proof. The lemma follows from Theorem 4 and Lemma 6. □

5.3. Updating data structures

When updating the data structures with a newly computed transition we must do it in such a way that the resulting automaton stays finite and the construction terminates.

The procedure $\mathsf{update}(q_s,\tilde{\ell},E)$ taking care of this is defined in Fig. 22, where the parameters q_s, $\tilde{\ell}$, and E denote a transition labelled $\tilde{\ell}$ from state q_s to a (potentially) new state characterised by E that is to be added to the automaton. The procedure does the following:

The line (1) first checks whether a suitable state is already present in the automaton. Here we enforce a partitioning of states according to the domains of their corresponding multisets of exposed prefixes – identifying two states, q_1 and q_2, if $\mathsf{dom}(\mathsf{E}[q_1]) = \mathsf{dom}(\mathsf{E}[q_2])$. As noted in [31] other partitionings are possible but we shall not go further into this. If a suitable state, q, exists it is used as the target state of the new transition in line (2), otherwise a fresh state, q_t, is inserted into the automaton and the corresponding entry in E initialised to $\bot_{\mathfrak{M}}$ in line (3).

Then in line (4) it is checked whether the multiset $\mathsf{E}[q_t]$ corresponding to the target state already includes the information contributed by E. If this is not the case the information is updated using the *widening operator*, $\nabla_{\mathfrak{M}} : \mathfrak{M} \times \mathfrak{M} \to \mathfrak{M}$ (see e.g. [23]), described below and the state is added to the worklist in line (6). The widening operator is defined by

$$(M_1 \nabla_{\mathfrak{M}} M_2)(\ell) = \begin{cases} M_1(\ell) & \text{if } M_2(\ell) \leq M_1(\ell) \\ M_2(\ell) & \text{if } M_1(\ell) = 0 \wedge M_2(\ell) > 0 \\ \infty & \text{otherwise} \end{cases}$$

and ensures that new information is added to the pre-existing in a manner that stabilises after finitely many iterations and still ensures that $M_1 \max_{\mathfrak{M}} M_2 \leq_{\mathfrak{M}} M_1 \nabla_{\mathfrak{M}} M_2$. We refer to [31] for the proof.

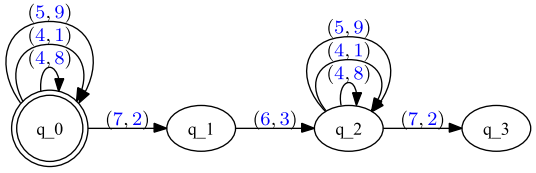
In line (7) the new transition $(q_s,\tilde{\ell},q_t)$ is added to the automaton while all of the pre-existing transitions out of q_s along $\tilde{\ell}$ are removed as their destination may no longer be correct. This may leave some states unreachable and in line (8) a suitable clean-up procedure, defined below, is invoked in order to ensure that these are removed.

Cleaning up

The cleanup procedure shown in Fig. 23 simply computes the set of states reachable from the start state $q_\star$ and uses this to

(i) restrict the set of states, Q, and the set of transitions, δ, by intersection
(ii) clean out the worklist, W, by intersection

Example 7. Applying the pathway analysis to P_{eat} we obtain the result shown below. Comparing to Example 2 this corresponds to the exact behaviour that arises from collapsing frames 1, 2, and 3 into one and 5, 6, and 7 into one. In each of these cases it is not possible to distinguish the states corresponding to the collapsed frames because the global multisets of exposed reaction capabilities are the same.



5.4. Correctness of the algorithm

Intuitively the outlined worklist algorithm is correct if it terminates producing a finite partially deterministic automaton able to faithfully simulate all transition sequences of the program, $P_\star$, of interest.

We address these issues separately starting with termination, where the following result holds:

Lemma 14 (Termination). *The worklist algorithm always terminates.*

Proof. Clearly, for any program, $P_\star$, the algorithm operates over a finite set, $\mathbf{Lab}_\star$, of labels. Now let us consider a possibly non-terminating execution. Note that Q as well as $\mathsf{E}[\cdot]$ grow in a non-decreasing manner.

The set $\{\mathsf{dom}(\mathsf{E}[q]) \mid q \in \mathsf{Q}\}$ grows in a non-decreasing manner and since $\mathsf{dom}(\mathsf{E}) \subseteq \mathbf{Lab}_\star$ the value of the set must eventually stabilise. After this the test in line (1) of Fig. 22 will always succeed and the production of new states in line (3) will cease. Thus Q stabilises.

The vector $(\mathsf{E}[q])_{q\in\mathsf{Q}}$ grows in a non-decreasing manner but due to the properties of the widening it must eventually stabilise thereby stopping the growth of W.

From this point on lines (4–7) of Fig. 21 will decrease the size of W by one in every iteration. Eventually W will be empty and hence the algorithm will terminate. □

We then turn to the correctness of the format of the output. Here we have:

Lemma 15. *The worklist algorithm always produces a partially deterministic automaton.*

Proof. We prove the claim by showing that

$$\forall(q_1,\tilde{\ell}_1,q_1'),(q_2,\tilde{\ell}_2,q_2') \in \delta : q_1 = q_2 \wedge \tilde{\ell}_1 = \tilde{\ell}_2 \Rightarrow q_1' = q_2'$$

is an invariant at line (4) of Fig. 21. It is maintained due to the construction of δ in line (6) of Fig. 22. □

Finally, we address the correctness of the contents of the output. We will show this as a simulation result. We shall say that a state q denoting the exposed capabilities E represents a process P whenever $P \triangleright E$ where

$$P \triangleright E \quad \text{iff } \mathcal{E}_\star[\![P]\!] \preceq_{\mathfrak{R}} E$$

Using this we can state:

Lemma 16. *If* $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $P \Rightarrow Q$ *then* $P \triangleright E$ *if, and only if,* $Q \triangleright E$.

Proof. This follows from Lemma 6. □

Now the following result shows that a single step in the semantics is correctly simulated by the automaton:

Theorem 17. *The worklist algorithm produces a finite automaton,* $(\mathsf{Q},q_\star,\delta,\mathsf{E})$, *such that, if*

$P \triangleright \mathsf{E}[q]$ *and* $P \xrightarrow{\tilde{\ell}} Q$,

$$\begin{aligned} Q_{reach} &:= \{q_\star\} \cup \{q \mid \exists n, \exists q_1,\cdots,q_n : (q_\star,\cdots,q_1) \in \delta \wedge \cdots \wedge (q_n,\cdots,q) \in \delta\};\\ \mathsf{Q} &:= \mathsf{Q} \cap Q_{reach};\\ \delta &:= \delta \cap (Q_{reach} \times (\mathbf{Lab} \times \mathbf{Lab}) \times Q_{reach})\\ \mathsf{W} &:= \mathsf{W} \cap Q_{reach}; \end{aligned}$$

Fig. 23. clean-up(Q,W,δ).

there exists a unique $q' \in \mathsf{Q}$ *such that*

$$Q \triangleright \mathsf{E}[q'] \text{ and } (q,\tilde{\ell},q') \in \delta.$$

Proof. Consider the last time, t_0, that the state q was removed from W in line (4) of Fig. 21. Now let E_0 denote the corresponding values of the data structures such that $\mathsf{E}_0[q] = \mathsf{E}[q]$ and hence $P \triangleright \mathsf{E}_0[q]$.

It follows from $P \xrightarrow{\tilde{\ell}} Q$, Lemma 13, and the fact that enabled is monotonic that $\tilde{\ell} \in \mathsf{enabled}(\mathsf{E}_0[q])$ and hence that $\tilde{\ell}$ is selected for consideration in line (5) of Fig. 21. By Theorem 45 line (6) then produces E such that $Q \triangleright E$.

Following line (7) of Fig. 21 it is immediate that lines (1–3) in Fig. 22 identify a state, q', and that lines (4–6) yield $(q,\tilde{\ell},q') \in \delta_1$ and $E \leq_{\mathfrak{M}} \mathsf{E}_1[q']$, where δ_1 and E_1 denote the corresponding new data structures.

As this is the last iteration over q there will be no further calls of $\mathsf{update}(q,\tilde{\ell},\ldots)$. Thus line (8) of Fig. 23 will not remove $(q,\tilde{\ell},q')$ from δ at a later stage. Clearly the values of $\mathsf{E}[\cdot]$ grow in a non-decreasing manner and, writing δ and E for the final values of the data structures, we have $(q,\tilde{\ell},q')$ and $E \leq_{\mathfrak{M}} \mathsf{E}_1[q'] \leq_{\mathfrak{M}} \mathsf{E}[q']$, which completes the proof.

Uniqueness of q' follows from Lemma 15. □

We may define the reflexive and transitive closure of δ inductively as follows:

$$(q_\bigstar,\varepsilon,q_\bigstar) \in \delta^\bigstar \quad \frac{(q_\bigstar,\Lambda,q) \in \delta^\bigstar \quad (q,\tilde{\ell},q') \in \delta}{(q_\bigstar,\Lambda\tilde{\ell},q')}$$

which allows us to state the following corollary:

Corollary 18. *The worklist algorithm produces a finite automaton,* $(\mathsf{Q},q_\bigstar,\delta,\mathsf{E})$*, such that, if*

$$P_\bigstar \xrightarrow{\Lambda}{}^\bigstar P,$$

there exists a $q \in \mathsf{Q}$ *such that*

$$P \triangleright \mathsf{E}[q] \text{ and } (q_\bigstar,\Lambda,q) \in \delta^\bigstar.$$

Proof. The proof follows straightforwardly by induction on the length of Λ. □

This shows that arbitrary reaction sequences are correctly simulated by the automaton.

we have implemented the algorithm and find that it performs well on the small-scale examples where we have used it – including the extended example of the next section. Its worst case complexity is unfortunately rather high – exponential time and space – essentially because the automaton in the worst case might contain a number of states that is exponential in the number of labels occurring in $P_\bigstar$.

6. A model of cholesterol uptake

In the following we shall model and investigate an *endocytic pathway* facilitating a biological process called *receptor mediated endocytosis*. This process is common in mammalian cells, where it is a general mechanism for subsuming particles from the blood stream.

The best known example of this process is the *LDL degradation pathway*, further discussed in Section 6.1. By this mechanism cells acquire the *cholesterol* required for the membrane synthesis that occurs during cell growth [19, Section 18.3][1, pp. 749–750]. It is also a common source of medical conditions as even small errors in the active components greatly increase the risk of cardiovascular disease. As we shall see in Section 6.3 our analysis is able to illustrate the more immediate effects of such component defects.

6.1. The LDL degradation pathway

Before turning to the modelling and subsequent analysis we shall briefly discuss the biological system of interest, as illustrated in Fig. 24.

Cholesterol is mainly obtained from *Low-density Lipoproteins* (LDLs), which carry cholesterol in the form of tightly packed *cholesteryl esters*.

Specialised *transmembranal receptor proteins* that perform free lateral diffusion in the *plasma membrane* of the cell recruit the LDLs from the blood. When the *extra-cellular* domain of such an *LDL receptor* encounters the *ApoB* domain exposed by an LDL particle the two particles will bind to each other by *complexation*.

Meanwhile, and independent of this, *clathrin particles* continuously assemble on the *cytosolic* side of the plasma membrane – thereby forcing it to form *clathrin coated pits* that grow progressively deeper until released into the cytosol as separate *clathrin coated vesicles*.

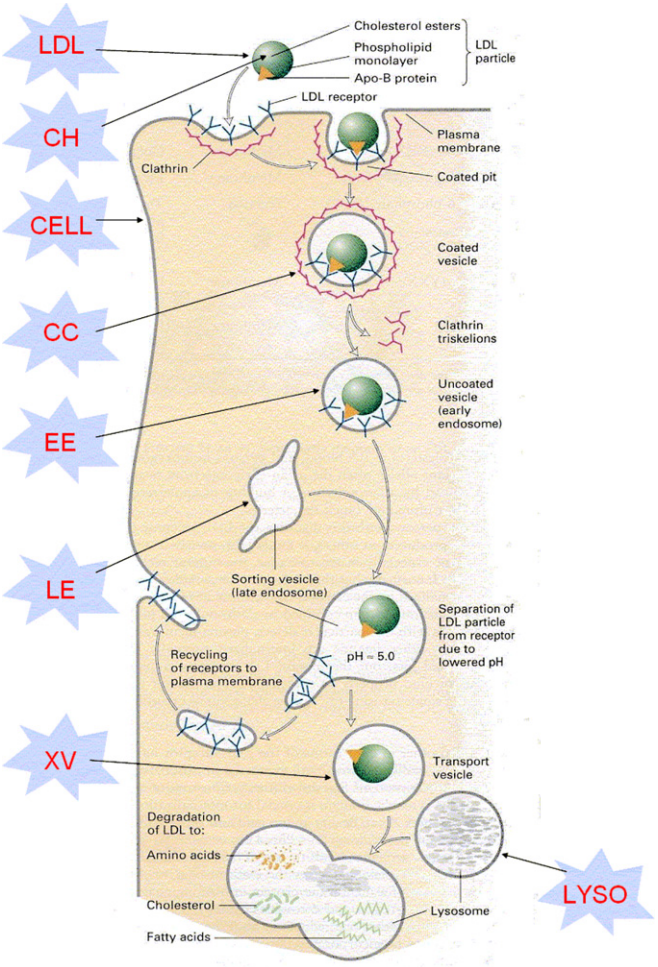


Fig. 24. The LDL cholesterol degradation process [19].

The diffusing receptors tend to associate with clathrin coated pits because their *intra-cellular* domain binds to complementary *adaptin* domains (molecules really) exposed by the clathrin coat. Such associated receptors and the LDLs that they bind, if any, are internalised when the coated vesicle is formed.

Once internalised, coated vesicles shred their clathrin coat and become *early endosomes*. At this stage the LDL/receptor complex is still intact. This changes, however, when the early endosome merges with a *late endosome*. The acidic environment in this compartment makes the receptors separate from the LDLs.

From the late endosomes the receptor proteins are recycled to the plasma membrane. The LDL molecules, however, are transferred by vesicles to *lysosomes* where they are finally hydrolysed in order to free the necessary cholesterol.

6.2. The BioAmbients model

In order to subject this biological system to formal analysis we have modelled it in the BioAmbients language as shown in Fig. 25 and explained below.

In accordance with Regev's examples and guidelines we take the approach that each kind of physical compartment as well as each kind of multiprotein complex should correspond to one ambient role. Recall that we indicate the roles of ambients by superscripted annotations, $\mu \in \mathbf{Role}$, i.e. we write $\lceil P \rceil^{\mu}$ for an ambient of role μ. The resulting model is shown in Fig. 25 and uses the following roles:

- The **CH** role models cholesterol (molecular compartments) in lines 1–2.
- The **LDL** role models LDL particles (molecular) in lines 3–5.
- The **EE** role models early endosomes (membrane-bound) in lines 6–9.
- The **CC** role models clathrin coats (molecular) in lines 10–11.
- The **XV** role models transfer vesicles (membrane-bound) in lines 12–13.

```
let
(1)  CHolesterol ≜
(2)      [ exit Hydr[8] ]CH
(3)  LowDensityLipoprotein ≜
(4)      [ (ApoB) LDLrcpt#!{ApoB}[1] . enter ApoB[2] . enter ee[3] . enter xv[4] .
(5)                syncXV^!{Le}[5] . proc^?{Hydr}[6] . (expel Hydr[7] | CHolesterol ]LDL
(6)  endo ≜
(7)      enter AP2[10,18,26] . syncCCEE^?{le}[11,19,27] . exit AP2[12,20,28] . merge– le[13,21,29]
(8)  EarlyEndo ≜
(9)      [ accept ee[17,25] . endo ]EE
(10) ClathrinCoat ≜
(11)     [ EErcpt?{ap2}[30] . accept ap2[31] . syncCCEE_!{Le}[32] . expel ap2[33] ]CC
(12) XferVesicle ≜
(13)     [ accept xv[36] . syncXV_?{le}[37] . exit le[38] . merge– lyso[39] ]XV
(14) LateEndo ≜
(15)     [ merge+ Le[34] . expel Le[35] | XferVesicle ]LE
(16) LYSOsome ≜
(17)     [ merge+ lyso[40] . proc_!{hydr}[41] ]LYSO
(18) Cell ≜
(19)     [ LDLrcpt#?{apob}[22] . accept apob[23] . EErcpt!{AP2}[24] . EarlyEndo
(20)       +EErcpt!{AP2}[14] . LDLrcpt#?{apob}[15] . accept apob[16] . EarlyEndo
(21)       +EErcpt!{AP2}[9] . [ endo ]EE
(22)     | ClathrinCoat
(23)     | LateEndo
(24)     | LYSOsome ]CELL
in
(25)     LowDensityLipoprotein | Cell
```

Fig. 25. The BioAmbients encoding the LDL degradation pathway.

- The **LE** role models late endosomes (membrane-bound) in lines 14–15.
- The **LYSO** role models lysosomes (membrane-bound) in lines 16–17.
- The **CELL** role models cells (membrane-bound) lines 18–24.

When we can do so without ambiguity, we will use the ambient roles also when referring to the biological entities that they model. As will be evident from the explanation below, the model emphasises the receptor dynamics that facilitates the initial binding of LDL but (for lack of space) ignores the details of receptor recycling. In Nature each compartment and reaction would be present in the thousands. The analyses we perform here, however, are qualitative rather than quantitative and therefore it suffices for us to model a single representative for each biological entity. Note that some primitive prefixes have more than one label as we assign a unique label for every distinct use of the defining macro; the macro endo, e.g., occurs in lines 19–21 and hence each prefix has three labels.

As evident from Fig. 24 the **LDL** (defined by LowDensityLipoprotein, lines 3–5) is initially located outside of the **CELL** (defined by Cell, lines 18–24) in the manner shown in Fig. 25 line 25. Here it offers an *ApoB* signal via the channel *LDLrcpt* that corresponds to the extra-cellular binding site of the transmembranal LDL receptor of the **CELL** (line 4, label 1).

At this stage the early endosome has not been formed yet. We model the transmembranal LDL receptors and the membrane patch that will later fold into the early endosome as a process capable of evolving into the **EE** ambient (lines 19–21). As explained, the clathrin coated early endosome may be formed with or without bound LDL particles. We model this as a non-deterministic external choice such that one of the following three binding scenarios may occur before the **EE** ambient is released:

(i) The extra-cellular part *LDLrcpt* of the LDL receptor binds the *ApoB* signal of the **LDL** (line 19, label 22 and line 4, label 1) thus forcing **LDL** to enter the **CELL** (line 4, label 2 and line 19, label 23). Subsequently the intra-cellular part *EErcpt* of the receptor is bound by the *AP*2 domain exposed by the **CC** bound adaptins (line 19, label 24 and line 11, label 30).

(ii) The intra-cellular part *EErcpt* of the receptor is bound by the *AP*2 domain exposed by the **CC** bound adaptin (line 20, label 14 and line 11, label 30). Subsequently the extra-cellular part *LDLrcpt* of the LDL receptor binds the *ApoB* signal of the **LDL** (line 20, label 15 and line 4, label 1) thus forcing **LDL** to enter the **CELL** (line 4, label 2 and line 20, label 16).

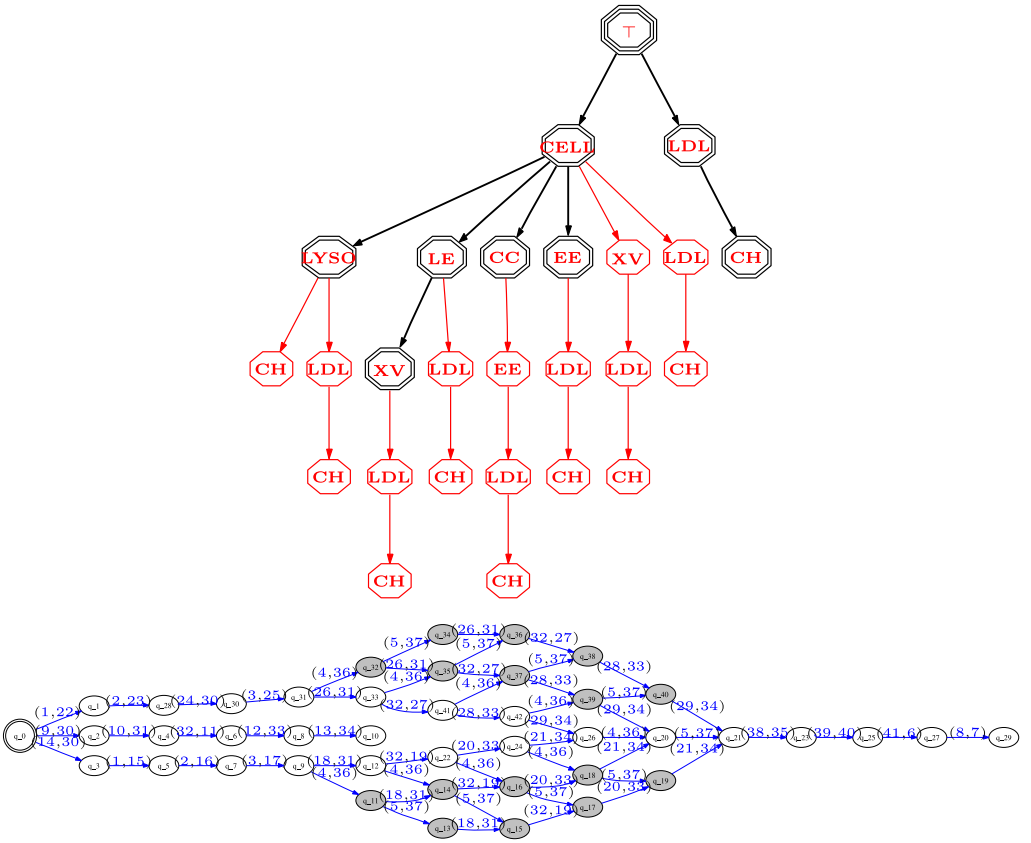


Fig. 26. Normal receptors.

(iii) The intra-cellular part *EErcpt* of the receptor is bound by the *AP2* domain exposed by the **CC** and the extra-cellular part *LDLrcpt* is never bound (line 21, label 9 and line 11, label 30).

Note that two choices have identical first prefixes. This is prohibited by the well-formedness condition of Regev et al [37], who are concerned with the computation of stochastic rates. We have no such concerns and therefore allow it.

If the **LDL** is in place inside the **CELL** after the binding scenario it may enter the **EE** (line 4, label 3 and line 9, label 17 or 25) otherwise not. Either way, the internalisation of the clathrin coated pit may be completed by the **EE** entering the **CC** (line 7, labels 10,18, or 26 and line 11, label 31).

In Nature this internalisation process is atomic since the $[[[[[\ ]^{\mathbf{CH}}]^{\mathbf{LDL}}]^{\mathbf{EE}}]^{\mathbf{CC}}]^{\mathbf{CELL}}$ (or $[[[\ ]^{\mathbf{EE}}]^{\mathbf{CC}}]^{\mathbf{CELL}}$) configuration arises instantaneously when the coated vesicle is completed and internalised. By modelling this as a sequence of events we introduce *modelling artifacts* as is indeed a common phenomenon when describing biological systems using process algebras. Most importantly, for the **LDL** to enter the **EE** we have to allow it into the **CELL**, which is biologically unsound. We must keep this in mind when interpreting the analysis results. Also the **EE** must pass the **CC** in order to enter the **CELL**. We enforce this by synchronising the **CC** and the **EE** via an exchange of the token *Le* on the channel *syncCCEE* (line 7, label 11,19, or 27 and line 11, label 32). Once synchronised the **EE** can freely leave the **CC** (line 7, label 12,20, or 28 and line 11, label 33). This corresponds to the internalised early endosome shredding its clathrin coat.

Knowing the token *Le* from the synchronisation the **EE** is now able to merge with the **LE** (line 7, label 13,21, or 29 and line 15, label 34). This releases the **LDL** into the **LE** from where it may enter the **XV** (line 4, label 4 and line 13, label 36). Once the **LDL** is inside the **XV** they are able to synchronise by reusing the token *Le* for an exchange on the channel *syncLDLXV* (line 5, label 5 and line 13, label 37). This synchronisation gives **XV** the ability to leave the **LE** taking the **LDL** cargo with it (line 13, label 38 and line 15, label 35).

Finally, the **XV** may merge with the **LYSO** (line 13, label 39 and line 17, label 40), thus releasing the **LDL** cargo into its final destination where it may be hydrolysed into **CH** (line 5, label 6 and line 17, label 41 followed by line 2, label 8 and line 5, label 7).

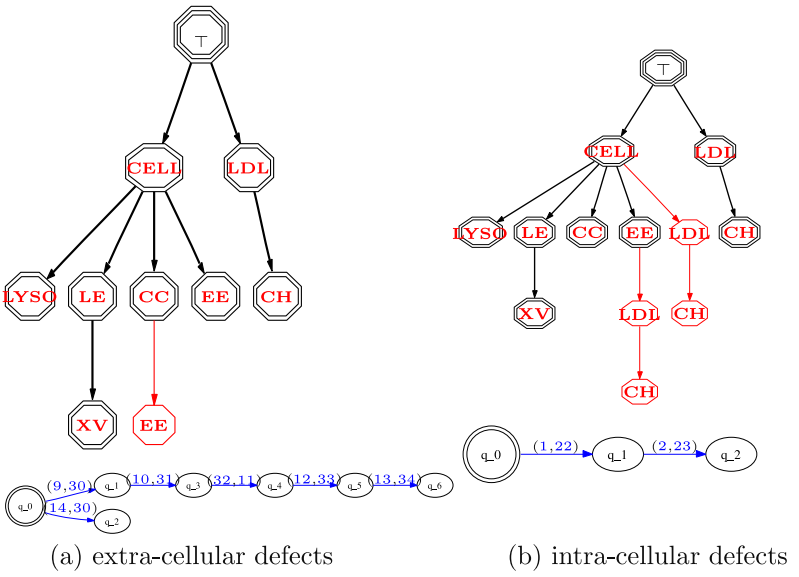


Fig. 27. Analysis results for defect systems.

6.3. Analysing the LDL degradation pathway

It is rather straightforward to improve the precision of the CFA by adding context information. We obtain a so-called 2CFA, where all information is localised, by always dereferencing the information collected by the analysis relations using the roles of the 2 immediately enclosing ambients. Experience indicates that this does not influence the pathway analysis, but that the computed control flow graphs are clearer and slightly more precise [35]. In the following we take advantage of this fact when presenting the analysis results for the LDL degradation pathway.

Thus, when subjecting the model to control flow analysis we obtain a result that can be represented graphically as shown in Fig. 26 (top). Except for the special $\top$ node with triple borders, which represents the super-environment, the nodes represent ambients and the edges represent the containment relation $\mathcal{I}$. The nodes with double borders connected with bold (black) edges represent the system in its initial state. The remaining (red)[1] nodes and edges account for the dynamic evolution of the system.

As shown in Fig. 26 the analysis reveals a system that largely behaves as expected; in particular we notice that **CH** may be released inside **LYSO**. Note that in the biological system the **LDL** is never able to float freely in the cytosol (the top-level fluid of the **CELL**); as mentioned in Section 6.2 the occurrence in the figure is caused by a modelling artifact. Also note that the **LDL** *can* occur inside an un-coated **EE** after the **CC** has been shredded; in this case the double bordered **EE** represents both the initial configuration and a later stage of evolution.

When subjecting the model to the pathway analysis we obtain the automaton shown in Fig. 26 (bottom). The resulting automaton clearly identifies the transitions that lead to the configurations identified by the CFA. The gray nodes, however, show *analysis artifacts*, i.e. states that are not reachable in practice. In the upper part of the automaton the transition (4,36) cannot occur before (29,34) because the **EE** must merge with the **LE** before **LDL** can enter the **XV**. For similar reasons (4,36) cannot happen before (21,34) in the bottom part. Given the nature of approximative techniques it is clear that artifacts are to be expected. We conjecture, however, that a pathway analysis enhanced with spatial information would not exhibit these particular artifacts.

Disorders. Some mammals suffer from the inherited disorder *familial hypercholesterolemia*, which dramatically increases the risk of the cardiovascular disease *atherosclerosis*. This disorder is caused by defects in the LDL receptor proteins that originate from inherited mutations.

When such a defect is located in the extra-cellular part of the receptor it is no longer able to bind an LDL particle. We model this phenomenon simply by introducing a spelling mistake in the receiving end of the *LDLrcpt* channel, i.e. (*LDLrcp*^?{*apob*}, *LDLrcpt*_!{*ApoB*}). As can be seen from Fig. 27a (top) the CFA reveals that the cell can no longer internalise LDL particles. It still internalises early endosomes but only empty ones (**EE** may occur inside **CC** within **CELL** but **EE** carries no **LDL** cargo).

Indeed, as revealed by the corresponding pathway analysis Fig. 27a (bottom), the internalisation of **EE** still works completely as intended. Only the extracellular binding capacity is affected.

[1] For interpretation of color in Fig. 26 and Fig. 27, the reader is referred to the web version of this article.

Finally, when the defect is located in the intra-cellular part of the receptor protein it can bind but not internalise an LDL particle. Again we model this by introducing a spelling mistake in the sending end of the *EErcpt* channel, i.e. (*EErcpt*?{*ap*2},*EErcp*!{*AP*2}). The CFA result shown in Fig. 27b (top) shows the effect: **LDL** may occur inside **EE** but **EE** never enters **CC**. Again, the **LDL** floating freely in the cytosol (**CELL**) is caused by a modelling artifact.

More interestingly, however, the pathway analysis clearly shows that no reaction allowing **LDL** to enter **EE** can ever take place. This shows that the two analysis approaches are indeed complementary and indicates an iterative analysis strategy: Just as positive information is propagated from the CFA to the pathway analysis in terms of $\mathscr{F}_\star$, negative information can be propagated back into the CFA in order to reduce $\mathscr{F}_\star$. Iterating this process improves the results of both the CFA and the pathway analysis [33].

In biological terms the results indicate that the system, as modelled here, cannot perform its normal function if the receptors are somehow defective. At this level of interpretation the results coincide completely with biological reality. Since the proposed analyses are efficient this leads us to hypothesise that this approach can be used as a first means to identify pathways affected by this kind of system perturbations.

7. Conclusion

Most static analyses previously developed for the BioAmbients modelling language have been concerned with properties of the *configurations* that might be reachable during evaluation. As previously shown in [32,35] this provides a sound platform for the investigation of the *spatial properties* of models.

In contrast, the *pathway analysis* defined in this paper focuses on the sequences of *transitions* that might be realised during evaluation. The resulting analysis provides a firm basis for the investigation of the *causal properties* of models.

The study of the LDL Degradation Pathway in Section 6 illustrates that the two types of information collected by the analyses are complementary in a manner that allows an effective scheme of mutual iteration to be fashioned. It turns out that such a scheme improves the precision of both analyses [33].

In technical terms the presented analysis computes a finite partially deterministic automaton that over-approximates the full set of, possibly infinite, sequential behaviours realisable by a given model. The resulting automaton is *safe* in the sense that it faithfully embeds the causal orderings that govern the possible evolutions of the model.

This approximation was achieved by combining elements from Flow Logic based control flow analysis, classical Monotone Frameworks, and Abstract Interpretation. First we modelled the configurations of the system by extended multisets of exposed reactive prefixes. In order to capture the dynamic nature of processes we then performed a detailed analysis of how the extended multisets grow and shrink as prefixes react. In the style of classical bit vector frameworks this information was used to specify transfer functions describing how the extended multisets evolve as reactions happen. Finally we used a classical worklist algorithm for computing the desired automata. Termination of this algorithm was ensured by the use of widening.

The practical value of this approach is in part determined by its computational complexity. The presented algorithm is remarkably flexible in this respect. A fine grained equivalence relation on states is likely to give a fairly precise analysis result at a high complexity. In contrast a coarser equivalence relation will give a more approximative analysis result at a lower complexity. It is worth mentioning at this point that working implementations are already in place and that the analysis examples presented throughout this paper have all been automatically produced.

We are confident that the presented analysis provides a strong tool for studying the (causal) properties of biological systems. While unable to *ensure* that given reactions take place the established over-approximation is still immensely useful in demonstrating that undesirable (or indeed desirable) reactions, perhaps part of a dysfunctional pathway, cannot arise. As noted in the introduction the range of potential applications is wide, ranging from simple modelling support (e.g. in the flavour of debugging) to actual predictions (e.g. in the flavour of in-silico screening of drug delivery mechanisms).

In future work we may extend the present approach into a simultaneously over- and under-approximating analysis. We conjecture that this would be a good platform for addressing stochastic issues. Another avenue to be explored is a refinement of the analysis where spatial information is added to the states of the computed automata. We intend to incorporate symbolic techniques from the field of model checking [9] in order to improve on the exponential space worst case complexity exhibited by the present formulation. Finally, a data-mining facility based on logical languages could easily be implemented on top of the computed graphs and automata.

Appendix

A. Proofs of properties of programs

Here we shall be concerned with proving that the property of being a program is preserved under semantic evaluation. First we shall show a number of minor results.

We start by showing some basic properties of free names:

Fact 19. Assume $Q = \mathrm{rec}\,X.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$\begin{aligned}\mathsf{fn}_\Gamma(P[^Q/_X]) &= \mathsf{fn}_{\Gamma[X\mapsto \mathsf{fn}_\Gamma(Q)]}(P)\\ &= \begin{cases}\mathsf{fn}_{\Gamma[X\mapsto\emptyset]}(P) \cup \mathsf{fn}_\Gamma(Q) & \text{if } X \in \mathsf{fpi}(P)\\ \mathsf{fn}_\Gamma(P) & \text{otherwise}\end{cases}\end{aligned}$$

Proof. The result follows by structural induction on P. In the case of restriction we note that α-renaming ensures that the names bound in P cannot capture the free names of Q. A similar property is ensured for summations as the well-formedness condition demands that $X \in \mathsf{fpi}(P) \Rightarrow (\lfloor \mathsf{bn}(M_i)\rfloor \cap \lfloor \mathsf{fn}(Q)\rfloor = \emptyset)$ for all i, where we note that $\mathsf{fpi}(Q) = \emptyset \Rightarrow (\mathsf{fn}_\Gamma(Q) = \mathsf{fn}(Q))$ for all Γ. □

Fact 20. $\mathsf{fn}_\Gamma(P[^m/_x]) = \begin{cases}(\mathsf{fn}_\Gamma(P) \setminus \{x\}) \cup \{m\} & \text{if } x \in \mathsf{fn}_\Gamma(P)\\ \mathsf{fn}_\Gamma(P) & \text{otherwise}\end{cases}$

Proof. By structural induction on P. □

Fact 21. If $\mathsf{fpi}(P) = \emptyset$ and $\mathbf{C} \vdash P$ then both of the following hold:

(i) If $P \Rrightarrow Q$ then $\mathsf{fn}(P) = \mathsf{fn}(Q)$.

(ii) If $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{fn}(P) \supseteq \mathsf{fn}(Q)$.

Proof. The proof of (1) proceeds by induction on the inference of $P \Rrightarrow Q$. In the case of H-ALPH we use Fact 20. In the case of H-UREC we use Fact 19.

The proof of (2) follows by induction on the inference of $P \xrightarrow{\tilde{\ell}} Q$, where we use Fact 20 for the communication axioms and (1) in conjunction with the induction hypothesis in the case of R-AUX. □

We establish a similar result for the free process identifiers:

Fact 22. If $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

(i) If $P \Rrightarrow Q$ then $\mathsf{fpi}(Q) = \emptyset$.

(ii) If $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{fpi}(Q) = \emptyset$.

Proof. The proof of *(1)* follows by induction on the inference of $P \Rrightarrow Q$.

The proof of *(2)* follows by straightforward induction on the inference of $P \xrightarrow{\tilde{\ell}} Q$ using *(1)* in the case of R-AUX. □

A similar development is required for the higher level notion of well-formedness with respect to $\mathbf{C}$:

Fact 23. Assume $Q = \mathrm{rec}\,X.\,Q'$ and $\mathsf{fpi}(Q) = \emptyset$; if furthermore $P \prec Q$ then

$$\mathbf{C} \vdash_\Gamma P[^Q/_X] \Leftrightarrow \begin{cases}\mathbf{C} \vdash_{\Gamma'} P \wedge \mathbf{C} \vdash_\Gamma Q & \text{if } X \in \mathsf{fpi}(P)\\ \mathbf{C} \vdash_\Gamma P & \text{otherwise}\end{cases}$$

where $\Gamma' = \Gamma[X \mapsto \mathsf{fn}_{\Gamma[X\mapsto\emptyset]}(Q)]$.

Proof. The direction of $\Rightarrow$ follows by structural induction on P. All but two cases are straightforward.

In the case of name restriction we rely on Fact 25-*(1)*, the induction hypothesis, and the fact that α-conversion is disciplined.

The most interesting case is that of the recursive process. Here we have

$$\mathbf{C} \vdash_\Gamma (\mathrm{rec}\,Y.\,P)[^Q/_X]$$

and must show that, if $X \in \mathsf{fpi}(\mathrm{rec}\,Y.\,P)$ then

$$\mathbf{C} \vdash_{\Gamma'} \mathrm{rec}\,Y.\,P \text{ and} \quad (1)$$

$$\mathbf{C} \vdash_\Gamma Q \quad (2)$$

where $\Gamma' = \Gamma[X \mapsto \mathsf{fn}_{\Gamma[X\mapsto\emptyset]}(Q)]$, and $\mathbf{C} \vdash_\Gamma \mathsf{rec}\,Y.\,P$ otherwise. The latter case is trivially true and we proceed to show the former, i.e. (1) and (2). We have that

$$\begin{aligned}
&\mathbf{C} \vdash_\Gamma (\mathsf{rec}\,Y.\,P)[^Q/_X] \Rightarrow \\
&\qquad \mathbf{C} \vdash_{\Gamma''} P[^Q/_X] && \text{(I)}\\
&\qquad \wedge\, X \in \mathsf{fpi}(P[^Q/_X]) && \text{(II)}\\
&\qquad \wedge\, (\forall M^\ell\,.\,P' \preceq P[^Q/_X] : Y \in \mathsf{fpi}(P') \Rightarrow && \text{(III)}\\
&\qquad\qquad \lfloor \mathsf{bn}(M) \rfloor \cap \lfloor \mathsf{fn}_\Gamma(\mathsf{rec}\,Y.\,(P[^Q/_X])) \rfloor = \emptyset)
\end{aligned}$$

where $\Gamma'' = \Gamma[Y \mapsto \mathsf{fn}_{\Gamma[Y\mapsto\emptyset]}(P[^Q/_X])]$.

Knowing that $X \in \mathsf{fpi}(P)$ we use the induction hypothesis on (I) to obtain

$$\begin{aligned}
&\mathbf{C} \vdash_{\Gamma''} P[^Q/_X] \Rightarrow \mathbf{C} \vdash_{\Gamma'''} P && \text{(IV)}\\
&\qquad\qquad \wedge\, \mathbf{C} \vdash_{\Gamma''} Q && \text{(V)}
\end{aligned}$$

where $\Gamma''' = \Gamma''[X \mapsto \mathsf{fn}_{\Gamma''[X\mapsto\emptyset]}(Q)]$. Given that $\mathsf{fpi}(Q) = \emptyset$ the goal (2) follows from (V) and only (1) remains.

Now we observe that

$$\begin{aligned}
&\mathbf{C} \vdash_{\Gamma''''} P && \text{(a)}\\
&\wedge\, X \in \mathsf{fpi}(P) && \text{(b)}\\
&\wedge \left(\begin{array}{l} \forall M^\ell\,.\,P' \preceq P : Y \in \mathsf{fpi}(P') \Rightarrow \\ \qquad \lfloor \mathsf{bn}(M) \rfloor \cap \lfloor \mathsf{fn}_{\Gamma'}(\mathsf{rec}\,Y.\,P) \rfloor = \emptyset \end{array} \right) && \text{(c)}\\
&\qquad\qquad \Rightarrow \mathbf{C} \vdash_{\Gamma'} (\mathsf{rec}\,Y.\,P) && \text{(1)}
\end{aligned}$$

where $\Gamma'''' = \Gamma'[Y \mapsto \mathsf{fn}_{\Gamma'[Y\mapsto\emptyset]}(P)]$. Given that $\mathsf{fpi}(Q) = \emptyset$ the goal (b) follows from (II) and only (a) and (c) remain.

From (III) we get

$$\begin{aligned}
&(\forall M^\ell\,.\,P' \preceq P[^Q/_X] : Y \in \mathsf{fpi}(P') \Rightarrow \\
&\qquad \lfloor \mathsf{bn}(M) \rfloor \cap \lfloor \mathsf{fn}_\Gamma(\mathsf{rec}\,Y.\,(P[^Q/_X])) \rfloor = \emptyset)
\end{aligned}$$

Since $\mathsf{fpi}(Q) = \emptyset$ and according to the definition of Γ''' this is the same as

$$\begin{aligned}
&(\forall M^\ell\,.\,P' \preceq P : Y \in \mathsf{fpi}(P') \Rightarrow \\
&\qquad \lfloor \mathsf{bn}(M) \rfloor \cap \lfloor \mathsf{fn}_{\Gamma'''}(P) \rfloor = \emptyset)
\end{aligned}$$

However, it follows from simple calculations and Fact 19 that $\Gamma''' = \Gamma'''' = \mathsf{fn}_{\Gamma'}(\mathsf{rec}\,Y.\,P)$; hence (c) follows from (III) and (a) follows from (IV).

Finally, (1) follows from (a), (b), and (c), which concludes the proof of $\Rightarrow$.

The direction of $\Leftarrow$ follows by similar calculations. □

Fact 24. If $\mathbf{C} \vdash_\Gamma P$ and $\lfloor m \rfloor \in \mathbf{C}$ then $\mathbf{C} \vdash_\Gamma P[^m/_x]$

Proof. The proof proceeds by induction on the structure of P. In the case of restrictions we use that α-renaming is disciplined. In the case of recursive processes we rely on disciplined α-renaming to ensure that the separation between free and bound names is preserved by the substitution. The remaining cases are straightforward. □

Fact 25. If $\mathsf{fpi}(P) = \emptyset$ and $\mathbf{C} \vdash P$ then the following both hold:

(i) If $P \Rrightarrow Q$ then $\mathbf{C} \vdash Q$

(ii) If $P \xrightarrow{\tilde{\ell}} Q$ then $\mathbf{C} \vdash Q$

Proof. (1) follows by induction on the inference of $P \Rrightarrow Q$. In the case of H-ALPH we rely on the fact that α-renaming is disciplined. In the case of H-UREC we use Fact 23. In case of H-CAMB we use Fact 22-(1) in conjunction with the induction hypothesis. The remaining axioms are trivial and the remaining rules follow by the induction hypothesis.

(2) follows by induction on the inference of $P \xrightarrow{\tilde{\ell}} Q$. The axioms for movement yield the desired result by simple calculation whereas the axioms for communication use Fact 24. In the case of R-AMB we use Fact 22-(2) in conjunction with the induction hypothesis. Finally, R-AUX follows from *(1)*, and the remaining rules follow by application of the induction hypothesis. □

Lemma 26 (Subject reduction). *If* $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $P \xrightarrow{\tilde{\ell}} Q$ *then* $\mathsf{PRG}_{\mathbf{C}}(Q)$.

Proof. The result follows from Facts 25, 21, and 22. □

B. Proof of correctness for the CFA

Correctness, or *semantical soundness*, of the 0CFA requires that acceptability is preserved under reaction. In order to establish the required *subject reduction* result we rely on a number of minor results.

Correctness of $\mathsf{CP}_\star$ The result rests on a number of minor results:

Fact 27. Assume $Q = \mathsf{rec}\,X.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$\begin{aligned}\mathsf{prfx}_\Gamma(P[^Q/_X]) &= \mathsf{prfx}_{\Gamma[X\mapsto \mathsf{prfx}_\Gamma(Q)]}(P)\\ &= \begin{cases}\mathsf{prfx}_{\Gamma[X\mapsto\emptyset]}(P) \cup \mathsf{prfx}_\Gamma(Q) & \text{if } X \in \mathsf{fpi}(P)\\ \mathsf{prfx}_\Gamma(P) & \text{otherwise}\end{cases}\end{aligned}$$

Proof. The proof proceeds by induction in the structure of P. □

Facts 28. If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

(i) If $P \Rrightarrow Q$ then $\mathsf{prfx}_\Gamma(P) = \mathsf{prfx}_\Gamma(Q)$.

(ii) If $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{prfx}_\Gamma(P) \supseteq \mathsf{prfx}_\Gamma(.)Q$

Proof. For (1) the proof proceeds by induction on the shape of the proof tree establishing $P \Rrightarrow Q$, using Fact 27 in the case of H-UREC. For (2) the proof proceeds by induction on the shape of the proof tree establishing $P \xrightarrow{\tilde{\ell}} Q$, using (1) to prove the property in the case of R-AUX. □

Fact 29. Assume $Q = \mathsf{rec}\,X.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$\mathsf{CP}_{\Gamma\Delta}(P[^Q/_X]) = \begin{cases}\mathsf{CP}_{\Gamma'\Delta[X\mapsto\emptyset]}(P) \cup \mathsf{CP}_{\Gamma\Delta}(Q) & \text{if } X \in \mathsf{fpi}(P)\\ \mathsf{CP}_{\Gamma\Delta}(P) & \text{otherwise}\end{cases}$$

where $\Gamma' = \Gamma[X \mapsto \mathsf{prfx}_\Gamma(Q)]$.

Proof. The proof proceeds by structural induction on P. In the cases of parallel compositions and recursive processes it uses Fact 27. □

For (1) the proof proceeds by induction on the shape of the proof tree establishing $P \Rrightarrow Q$, using Fact 27 in the case of H-UREC. For (2) the proof proceeds by induction on the shape of the proof tree establishing $P \xrightarrow{\tilde{\ell}} Q$, using (1) to prove the property in the case of R-AUX.

Facts 30. If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

(i) If $P \Rrightarrow Q$ then $(\mathsf{CP}_{\Gamma\Delta}(P) = \mathsf{CP}_{\Gamma\Delta}(Q))$.

(ii) If $P \xrightarrow{\tilde{\ell}} Q$ then $(\mathsf{CP}_{\Gamma\Delta}(P) \supseteq \mathsf{CP}_{\Gamma\Delta}(Q))$.

Proof. For (1) the proof proceeds by induction on the shape of the proof tree establishing $P \Rrightarrow Q$, using Fact 29 in the case of H-UREC.

For (2) the proof proceeds by induction in the shape of the proof tree establishing $P \xrightarrow{\tilde{\ell}} Q$, using (1) in the case of R-AUX. □

Corollary 31 (Subject reduction)

If $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $P \xrightarrow{\tilde{\ell}} Q$ *then* $\mathsf{CP}(P) \supseteq \mathsf{CP}(Q)$.

Correctness of $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P$ We are now ready to show that the acceptability of analysis estimates is preserved under semantic reaction.

We shall find use for a minor fact regarding substitution and $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P$.

Fact 32. Assume $Q = \mathrm{rec}\, X.\, Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P[^{Q}/_{X}] \Leftrightarrow \begin{cases} (\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P \wedge (\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} Q & \text{if } X \in \mathsf{fpi}(P) \\ (\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P & \text{otherwise} \end{cases}$$

Proof. The fact follows by structural induction on P, where the well-formedness of P ensures that Q can only occur within μ. □

Now we can easily show that analysis acceptability is preserved under heating.

Lemma 33. *If* $\mathbf{C} \vdash P$ *and* $fpi(P) = \emptyset$ *then the following holds:*

If $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P$ *and* $P \Rrightarrow Q$, *we have* $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} Q$.

Proof. The proof proceeds by induction in the shape of the proof tree establishing $P \Rrightarrow Q$. In the case of *scope rules for name bindings* the lemma trivially holds because name restrictions are ignored by the analysis judgement. For the *congruence requirements* the result follows from the induction hypothesis. In the case of α*-equivalence* we have that if $P \equiv_{\alpha} Q$ then $\lfloor P \rfloor = \lfloor Q \rfloor$ and hence $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} \lfloor P \rfloor \Leftrightarrow (\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} \lfloor Q \rfloor$ follows by referential transparency. Finally, in the case of *unfolding of recursion* the desired result follows directly from Fact 32. □

To finally show the invariance under reaction we shall introduce an expansion of $\mathscr{I}$ into the new relation $\mathscr{I}@\mathscr{R}$, which takes into account the bindings of variables specified by the $\mathscr{R}$ component. This relation is defined as follows:

$$\text{If } M^{\ell} \in \mathscr{I}(\mu),\ x \in \mathsf{fn}(M) \text{ and } n \in \langle\mathscr{R}\rangle(x) \text{ then } M^{\ell}[^{n}/_{x}] \in \mathscr{I}@\mathscr{R}(\mu).$$

where we remind that the involved names are, indeed, canonical.

This expansion satisfies a useful substitution property.

Fact 34. If $\lfloor n \rfloor \in \mathscr{R}(x)$ and $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P$ then $(\mathscr{I}@\mathscr{F},\mathscr{R},\mathscr{F}) \models^{\mu} P[^{n}/_{x}]$.

Proof. The proof proceeds by structural induction on P. □

In this context we can show that the acceptability of analysis estimates is preserved under reaction in the following sense:

Lemma 35. *If* $\mathbf{C} \vdash P$ *and* $fpi(P) = \emptyset$ *then, if furthermore* $\mathsf{CP}(P) \subseteq \mathsf{CP}_{\bigstar}$ *the following holds:*

If $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P$ *and* $P \xrightarrow{\tilde{\ell}} Q$ *then* $(\mathscr{I}@\mathscr{R},\mathscr{R},\mathscr{F}) \models^{\mu} Q$ *and* $\tilde{\ell} \in \mathscr{F}$.

Proof. The proof is by induction on the inference of reactions $P \xrightarrow{\tilde{\ell}} Q$. In the case of movement it suffices to expand both $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\top} P$ and $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\top} Q$ using the definition of $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P$. In the case of communication we perform a similar expansion and then obtain the desired result using Fact 34. The remaining cases follow from the induction hypothesis and, in the case of R-AUX, uses Lemma 33. □

Corollary 36 (Subject reduction). *Assume* $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $\mathsf{CP}(P) \subseteq \mathsf{CP}_{\bigstar}$; *if furthermore* $(\mathscr{I},\mathscr{R},\mathscr{F}) \models^{\mu} P$ *and* $P \xrightarrow{\tilde{\ell}} Q$ *then* $(\mathscr{I}@\mathscr{R},\mathscr{R},\mathscr{F}) \models^{\mu} Q$ *and* $\tilde{\ell} \in \mathscr{F}$. □

This means that an analysis estimate that is acceptable for a process P is also acceptable for any process Q derived from it by a single reaction.

C. Proof of correctness for exposed prefixes

As was the case for the CFA correctness properties associated with the transfer functions of Section 4 relies on a number of minor results that we shall establish in the following.

Intuitively, *correctness of* $\mathcal{E}_{\Gamma}[\![P]\!]$ means *(i)* that it is invariant under heating and *(ii)* that it correctly captures the prefixes that may be involved in the first reaction step. We start by showing the usual substitution result:

Fact 37. Assume $Q = \mathsf{rec}\,X.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$\mathcal{E}_\Gamma[\![\, P[^Q/_X] \,]\!] = \mathcal{E}_{\Gamma[X \mapsto \mathcal{E}_\Gamma[\![\, Q \,]\!]}[\![\, P \,]\!]$$

Proof. This is easily shown by structural induction on P. □

And then we go on to show the main result:

Lemma 38. *If $\mathbf{C} \vdash P$ and $\mathit{fpi}(P) = \emptyset$ then both of the following hold:*
(i) *If $P \Rrightarrow Q$ then $\mathcal{E}_\Gamma[\![\, Q \,]\!] = \mathcal{E}_\Gamma[\![\, P \,]\!]$*
(ii) *If $P \xrightarrow{(\ell_1,\ell_2)} Q$ then $\ell_1 \in \mathit{dom}(\mathcal{E}_\Gamma[\![\, P \,]\!])$ and $\ell_2 \in \mathit{dom}(\mathcal{E}_\Gamma[\![\, P \,]\!])$*

Proof. The proof of the first part is by induction on the inference of $P \Rrightarrow Q$. Most cases are trivial. However, in the case of unfolding of recursion we use Fact 25, that $\mathit{LFP}(\lambda E.\ \mathcal{E}_{\Gamma[X \mapsto E]}[\![\, P \,]\!])$ is indeed a fixed point, and Fact 37.

The second part follows by induction on the inference of $P \xrightarrow{\tilde{\ell}} Q$. The result is immediate for the axioms and in the case of R-AUX we make use of the first part of the lemma. The remaining rules follow from the induction hypothesis. □

Due to the different role that $\mathcal{G}_{\Gamma\Delta}[\![\ ]\!]$ plays in the transfer function, the *correctness of* $\mathcal{G}_{\Gamma\Delta}[\![\, P \,]\!]$ is slightly more involved than was the case for $\mathcal{E}_\Gamma[\![\, P \,]\!]$. As usual, substitution possesses nice properties:

Fact 39. Assume $Q = \mathsf{rec}\,X.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$\mathcal{G}_{\Gamma\Delta}[\![\, P[^Q/_X] \,]\!] = \begin{cases} \mathcal{G}_{\Gamma[X \mapsto \mathcal{E}_\Gamma[\![\, Q \,]\!]]\Delta[X \mapsto \perp_{\mathfrak{T}}]}[\![\, P \,]\!] \max_{\mathfrak{T}} \mathcal{G}_{\Gamma\Delta}[\![\, Q \,]\!] & \text{if } X \in \mathsf{fpi}(P) \\ \mathcal{G}_{\Gamma[X \mapsto \mathcal{E}_\Gamma[\![\, Q \,]\!]]\Delta[X \mapsto \perp_{\mathfrak{T}}]}[\![\, P \,]\!] & \text{otherwise} \end{cases} \tag{C.1}$$

Proof. The result is shown by structural induction on P. □

And, surely, the safety of the approximation, $\mathcal{G}_{\Gamma\Delta}[\![\, P \,]\!]$, must be preserved under heating:

Lemma 40. *If $\mathbf{C} \vdash P$ and $\mathit{fpi}(P) = \emptyset$ then the following holds:*

If $P \Rrightarrow Q$ then $\mathcal{G}_{\Gamma\Delta}[\![\, Q \,]\!] = \mathcal{G}_{\Gamma\Delta}[\![\, P \,]\!]$.

Proof. The proof is by induction on the inference of $P \Rrightarrow Q$. When showing the lemma for H-UREC we use Fact 25, Fact 39, and that $\mathrm{LFP}(\lambda G.\ \mathcal{G}_{\Gamma[X \mapsto \mathcal{E}_\Gamma[\![\, \mathsf{rec}\,X.P \,]\!]]\Delta[X \mapsto G]}[\![\, P \,]\!])$ is indeed a fixed point. In the case of H-CSUM we use Lemma 38-(1).
The remaining axioms follow by simple calculations and the rules by the induction hypothesis. □

Finally, we must show that the safety of the approximation is preserved under reduction. This amounts to the following 'local' subject reduction result:

Lemma 41. *If $\mathbf{C} \vdash P$ and $\mathit{fpi}(P) = \emptyset$ then the following holds:*

If $P \xrightarrow{\tilde{\ell}} Q$ then $\mathcal{G}_{\Gamma\Delta}[\![\, Q \,]\!] \leq_{\mathfrak{T}} \mathcal{G}_{\Gamma\Delta}[\![\, P \,]\!]$.

Proof. The proof is by induction on the shape of the inference of $P \xrightarrow{\tilde{\ell}} Q$. The axioms follow by straightforward calculation using that $\mathcal{G}_{\Gamma\Delta}[\![\, P \,]\!] = \mathcal{G}_{\Gamma\Delta}[\![\, P[^n/_p] \,]\!]$ and $\mathcal{E}_\Gamma[\![\, P \,]\!] = \mathcal{E}_\Gamma[\![\, P[^n/_p] \,]\!]$ in the case of communication.
For the rule using the structural congruence we use Lemma 40. The remaining rules follow by the induction hypothesis. □

Besides of being an under- rather than an over-approximation, the *correctness of* $\mathcal{K}_\Delta[\![\, P \,]\!]$ is rather similar to that of $\mathcal{G}_{\Gamma\Delta}[\![\, P \,]\!]$. We have the usual substitution property:

Fact 42. Assume $Q = \mathsf{rec}\,X.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$\mathcal{K}_\Delta[\![\, P[^Q/_X] \,]\!] = \begin{cases} \mathcal{K}_{\Delta[X \mapsto \perp_{\mathfrak{T}}]}[\![\, P \,]\!] \min_{\mathfrak{T}} \mathcal{K}_\Delta[\![\, Q \,]\!] & \text{if } X \in \mathsf{fpi}(P) \\ \mathcal{K}_{\Delta[X \mapsto \perp_{\mathfrak{T}}]}[\![\, P \,]\!] & \text{otherwise} \end{cases}$$

Proof. The result follows by structural induction on P. □

Then we must ensure that the safety of the approximation is preserved by heating:

Lemma 43. *If* $\mathbf{C} \vdash P$ *and* $fpi(P) = \emptyset$ *then the following holds:*

If $P \Rrightarrow Q$ *then* $\mathcal{K}_\Delta[\![P]\!] = \mathcal{K}_\Delta[\![Q]\!]$.

Proof. The proof is by induction on the inference of $P \Rrightarrow Q$. In the case of H-UREC we use Fact 25, that $\mathrm{LFP}(\lambda K.\ \mathcal{K}_{\Delta[X \mapsto K]}[\![P]\!])$ is indeed a fixed point, and Fact 42. The remaining axioms follow by simple calculations and the rules by the induction hypothesis. □

Also, we must show that the safety of the approximation is preserved under reduction:

Lemma 44. *If* $\mathbf{C} \vdash P$ *and* $fpi(P) = \emptyset$ *then the following holds:*

If $P \xrightarrow{\tilde{\ell}} Q$ *then* $\mathcal{K}_\Delta[\![P]\!] \leq_{\mathfrak{T}} \mathcal{K}_\Delta[\![Q]\!]$.

Proof. The proof is by induction on the shape of the inference of $P \xrightarrow{\tilde{\ell}} Q$. The axioms follow by straightforward calculation using that $\mathcal{K}_\Delta[\![P]\!] = \mathcal{K}_\Delta[\![P[^n/_p]]\!]$ in the case of communication.

In the case of R-AUX we use Lemma 43. The remaining rules follow by the induction hypothesis. □

The following result states that this transfer function provides a safe approximation to the exposed actions of the process that results from the transition:

Theorem 45 (Subject reduction). *If* $\mathbf{C} \vdash P$ *and* $fpi(P) = \emptyset$ *then the following holds:*

If $P \xrightarrow{\tilde{\ell}} Q$ *then* $(\mathcal{E}_\star[\![Q]\!] \leq_{\mathfrak{M}} \mathsf{transfer}_{P,\tilde{\ell}}(\mathcal{E}_\star[\![P]\!]))$.

Proof. The proof is by induction of the inference of $P \xrightarrow{\tilde{\ell}} Q$. We have the following cases:

Case R-ENT:
Writing *lhs* for

$$\left[\!\left[(\mathsf{enter}\ n^{\ell_1}\ .\ P + P') \,\middle|\, P'' \right]\!\right]^{\mu_1} \left|\left[\!\left[(\mathsf{accept}\ n^{\ell_2}\ .\ Q + Q') \,\middle|\, Q'' \right]\!\right]^{\mu_2}\right.$$

we observe that

$$\mathcal{E}_\star[\![P]\!] \leq_{\mathfrak{M}} \mathcal{G}_\star[\![\mathsf{enter}\ n^{\ell_1}\ .\ P + P']\!](\ell_1) \leq_{\mathfrak{M}} \mathcal{G}_\star[\![lhs]\!](\ell_1)$$

and

$$\mathcal{E}_\star[\![Q]\!] \leq_{\mathfrak{M}} \mathcal{G}_\star[\![\mathsf{accept}\ n^{\ell_2}\ .\ Q + Q']\!](\ell_2) \leq_{\mathfrak{M}} \mathcal{G}_\star[\![lhs]\!](\ell_2).$$

Similarly we have

$$\mathcal{K}_\star[\![lhs]\!](\ell_1) \leq_{\mathfrak{M}} \mathcal{K}_\star[\![\mathsf{enter}\ n^{\ell_1}\ .\ P + P']\!](\ell_1) \leq_{\mathfrak{M}} \mathcal{E}_\star[\![\mathsf{enter}\ n^{\ell_1}\ .\ P + P']\!]$$

and

$$\mathcal{K}_\star[\![lhs]\!](\ell_2) \leq_{\mathfrak{M}} \mathcal{K}_\star[\![\mathsf{accept}\ n^{\ell_2}\ .\ Q + Q']\!](\ell_2) \leq_{\mathfrak{M}} \mathcal{E}_\star[\![\mathsf{accept}\ n^{\ell_2}\ .\ Q + Q']\!].$$

By calculation we get

$$\begin{aligned}
&\mathcal{E}_\star[\![P]\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![P'']\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![Q]\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![Q'']\!] \\
&\quad \leq_{\mathfrak{M}} (\mathcal{E}_\star[\![\mathsf{enter}\ n^{\ell_1}\ .\ P + P']\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![P'']\!] -_{\mathfrak{M}} \mathcal{E}_\star[\![\mathsf{enter}\ n^{\ell_1}\ .\ P + P']\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![P]\!]) \\
&\quad +_{\mathfrak{M}} (\mathcal{E}_\star[\![\mathsf{accept}\ n^{\ell_2}\ .\ Q + Q']\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![Q'']\!] -_{\mathfrak{M}} \mathcal{E}_\star[\![\mathsf{accept}\ n^{\ell_2}\ .\ Q + Q']\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![Q]\!]) \\
&\quad \leq_{\mathfrak{M}} (\mathcal{E}_\star[\![lhs]\!] -_{\mathfrak{M}} \mathcal{K}_\star[\![lhs]\!](\ell_1,\ell_2)) +_{\mathfrak{M}} \mathcal{G}_\star[\![lhs]\!](\ell_1,\ell_2)
\end{aligned} \tag{C.2}$$

which is exactly the desired result. The remaining axioms follow by similar reasoning.

Case R-RES:
In this case the proof follows from the induction hypothesis as names are ignored by the Pathway Analysis.

Case R-AMB:
Again the proof is by the induction hypothesis as ambients are ignored by the Pathway Analysis.

Case R-PAR:
It follows from the induction hypothesis that

$$\mathcal{E}_\star[\![Q]\!] \leq_{\mathfrak{M}} (\mathcal{E}_\star[\![P]\!] -_{\mathfrak{M}} \mathcal{K}_\star[\![P]\!](\tilde{\ell})) +_{\mathfrak{M}} \mathcal{G}_\star[\![P]\!](\tilde{\ell}).$$

Furthermore we have

$$\mathcal{K}_\star[\![P \,|\, R]\!](\tilde{\ell}) \leq_{\mathfrak{M}} \mathcal{K}_\star[\![P]\!](\tilde{\ell})$$

and

$$\mathcal{G}_\star[\![P]\!](\tilde{\ell}) \leq_{\mathfrak{M}} \mathcal{G}_\star[\![P \,|\, R]\!](\tilde{\ell})$$

so that

$$\begin{aligned}\mathcal{E}_\star[\![Q]\!] &\leq_{\mathfrak{M}} (\mathcal{E}_\star[\![P]\!] -_{\mathfrak{M}} \mathcal{K}_\star[\![P]\!](\tilde{\ell})) +_{\mathfrak{M}} \mathcal{G}_\star[\![P]\!](\tilde{\ell}) \\ &\leq_{\mathfrak{M}} (\mathcal{E}_\star[\![P]\!] -_{\mathfrak{M}} \mathcal{K}_\star[\![P \,|\, R]\!](\tilde{\ell})) +_{\mathfrak{M}} \mathcal{G}_\star[\![P \,|\, R]\!](\tilde{\ell})\end{aligned}$$

Thus we may calculate

$$\begin{aligned}\mathcal{E}_\star[\![Q \,|\, R]\!] &= \mathcal{E}_\star[\![Q]\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![R]\!] \\ &\leq_{\mathfrak{M}} (\mathcal{E}_\star[\![P]\!] -_{\mathfrak{M}} \mathcal{K}_\star[\![P \,|\, R]\!](\tilde{\ell})) +_{\mathfrak{M}} \mathcal{G}_\star[\![P \,|\, R]\!](\tilde{\ell}) +_{\mathfrak{M}} \mathcal{E}_\star[\![R]\!] \\ &= ((\mathcal{E}_\star[\![P]\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![R]\!]) -_{\mathfrak{M}} \mathcal{K}_\star[\![P \,|\, R]\!](\tilde{\ell})) +_{\mathfrak{M}} \mathcal{G}_\star[\![P \,|\, R]\!](\tilde{\ell}) \\ &= (\mathcal{E}_\star[\![P \,|\, R]\!] -_{\mathfrak{M}} \mathcal{K}_\star[\![P \,|\, R]\!](\tilde{\ell})) +_{\mathfrak{M}} \mathcal{G}_\star[\![P \,|\, R]\!](\tilde{\ell})\end{aligned}$$

which finishes the case.

Case R-AUX:
This case is straightforward because the safety of $\mathcal{E}_\star[\![\,]\!]$, $\mathcal{G}_\star[\![\,]\!]$, and $\mathcal{K}_\star[\![\,]\!]$ is preserved by heating due to Lemmas 38, 40, and 43. □

Corollary 46 (Subject reduction)
If $\mathsf{PRG}_{\mathbf{C}}(P)$ *and* $P \xrightarrow{\tilde{\ell}} Q$ *then* $\mathcal{E}_\star[\![Q]\!] \leq_{\mathfrak{M}} \mathsf{transfer}_{P,\tilde{\ell}}(\mathcal{E}_\star[\![P]\!])$.

Finally, we establish the semantic soundness of the transfer function as a straightforward corollary of the following theorem:

Theorem 47 (Semantic correctness)

$$P_\star \xrightarrow{\tilde{L}}{}^{\star} P \xrightarrow{\tilde{\ell}} Q \Rightarrow (\mathcal{E}_\star[\![Q]\!] \leq_{\mathfrak{M}} \mathsf{transfer}_{P_\star,\tilde{\ell}}(\mathcal{E}_\star[\![P]\!])).$$

Proof. This follows by induction on the length of $\tilde{L}$. The base case follows from Corollary 46. The inductive step is established using Corollaries 41, 44, 46, and 1. □

This shows that the approximation is safe for all reaction sequences that may arise from an initial program, $P_\star$.

References

[1] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, Peter Walter, Molecular Biology of The Cell, fourth ed., Garland Science, 2002.
[2] Gerard Berry, Gerard Boudol, The chemical abstract machine, POPL'90: Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press, New York, NY, USA, 1990, pp. 81–94.
[3] Ralf Blossey, Luca Cardelli, Andrew Phillips, A compositional approach to the stochastic dynamics of gene networks, Trans. Comput. Syst. Biol. 3939 (2006) 99–122.
[4] Muffy Calder, Stephen Gilmore, Jane Hillston, Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA, Transactions on Computational Systems Biology VII, Lecture Notes in Computer Science, vol. 4230, Springer, 2006, pp. 1–23.

[5] Luca Cardelli, Bioware languages, in: Andrew Herbert, Karen S. Jones (Eds.), Computer Systems: Theory, Technology, and Applications – A Tribute to Roger Needham, Monographs in Computer Science, Springer, 2004, pp. 59–65.
[6] Luca Cardelli, Brane calculi, in: Vincent Danos, Vincent Schachter (Eds.), Computational Methods in Systems Biology (CMSB'04), Lecture Notes in Computer Science, vol. 3082, Springer, 2005, pp. 257–280.
[7] Luca Cardelli, Andrew D. Gordon, Mobile ambients, Theor. Comput. Sci. 240 (1) (2000) 177–213.
[8] Marc Chiaverini, Vincent Danos, A core modeling language for the working molecular biologist, Proc. of CMSB'03, Lecture Notes in Computer Science, vol. 2602, Springer, 2003, pp. 166.
[9] Edmund M. Clarke, Orna Grumberg, Doron A. Peled, Model Checking, MIT Press, 2000.
[10] Vincent Danos, Cosimo Laneve, Core formal molecular biology, European Symposium on Programming (ESOP03), Lecture Notes in Computer Science, vol. 2618, Springer, 2004.
[11] Vincent Danos, Cosimo Laneve, Formal molecular biology, Theor. Comput. Sci. 325 (1) (2004) 69–110.
[12] Vincent Danos, Sylvain Pradalier. Projective brane calculus, in: Proceedings of Computational Methods in Systems Biology (CMSB'04), 2004.
[13] Suresh Jagannathan, Stephen Weeks, A unified treatment of flow analysis in higher-order languages, POPL'95: Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM Press, New York, NY, USA, 1995, pp. 393–407.
[14] Naaman Kam, David Harel, Hillel Kugler, Rami Marelly, Amir Pnueli, E. Jane Albert Hubbard, Michael J. Stern, Formal modeling of c. elegans development: a scenario based approach, Proceedings of CMSB 2003, Lecture Notes in Computer Science, vol. 2602, Springer, 2003, pp. 4–20.
[15] Kunihiko Kaneko, Life: An Introduction to Complex Systems Biology (Understanding Complex Systems), Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[16] Hiroaki Kitano, Systems biology: a brief overview, Science 295 (5560) (2002) 1662–1664.
[17] Céline Kuttler. Modeling bacterial gene expression in a stochastic pi-calculus with concurrent objects, PhD Thesis, University of Lille 1, 2007.
[18] Francesca Levi, Davide Sangiorgi, Mobile safe ambients, ACM Transactions on Programming Languages and Systems (TOPLAS'03) 25 (1) (2003) 1–69.
[19] Harvey Lodish, Arnold Berk, S. Lawrence Zipursky, Paul Matsudaira, David Baltimore, James E. Darnell, Molecular Cell Biology, fourth ed., W.H. Freeman and Company, 1999, Fig. 24 reprinted with the permission of the publisher.
[20] Robin Milner, Communication and Concurrency, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
[21] Robin Milner, Communicating and Mobile Systems: The π-Calculus, Cambridge University Press, 1999.
[22] Flemming Nielson, Hanne Riis Nielson, Flow Logic: a multi-paradigmatic approach to static analysis, The Essense of Computation: Complexity, Analysis, Transformation, Lecture Notes in Computer Science, vol. 2566, Springer, 2002, pp. 223–244.
[23] Flemming Nielson, Hanne Riis Nielson, Chris Hankin, Principles of Program Analysis, Springer, 1999.
[24] Flemming Nielson, Hanne Riis Nielson, Henrik Pilegaard, What is a free name in a process algebra?, Inform. Process. Lett. 103 (5) (2007) 188–194.
[25] Flemming Nielson, Hanne Riis Nielson, Corrado Priami, Debora S. da Rosa. Control flow analysis for BioAmbients, in: Proceedings of the First Workshop on Concurrent Models in Molecular Biology (BioConcur 2003), Electronic Notes in Theoretical Computer Science, vol. 180, 2007, pp. 65–79.
[26] Flemming Nielson, Hanne Riis Nielson, Helmuth Seidl, A succinct solver for ALFP, Nordic J. Comput. 9 (2002) 335–372.
[27] Flemming Nielson, Hanne Riis Nielson, Hongyan Sun, Mikael Buchholtz, René Rydhof Hansen, Henrik Pilegaard, Helmuth Seidl, The succinct solver suite, Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04), Lecture Notes in Computer Science, vol. 2988, Springer, 2004, pp. 251–265.
[28] Hanne Riis Nielson, Flemming Nielson, Infinitary control flow analysis: a collecting semantics for closure analysis, in: POPL'1997: Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1997, pp. 332–345.
[29] Hanne Riis Nielson, Flemming Nielson, Flow logics for constraint based analysis, The Seventh International Conference on Compiler Construction (CC'98), Lecture Notes in Computer Science, vol. 1383, Springer, 1998, pp. 109–127.
[30] Hanne Riis Nielson, Flemming Nielson, Data flow analysis for CCS, in: Thomas Reps, Mooly Sagiv (Eds.), Festschrift for Reinhard Wilhelm, Lecture Notes in Computer Science, vol. 4444, Springer, 2006.
[31] Hanne Riis Nielson, Flemming Nielson, A monotone framework for CCS, Comput. Lang. Syst. Struct. (2008) (under revision).
[32] Hanne Riis Nielson, Flemming Nielson, Henrik Pilegaard. Spatial analysis of BioAmbients, in: SAS'04 11th, Lecture Notes in Computer Science, vol. 3148, 2004, pp. 69–83.
[33] Henrik Pilegaard, Language based techniques for systems biology, PhD Thesis, The Technical University of Denmark, 2007.
[34] Henrik Pilegaard, Flemming Nielson, H.R. Nielson, Static analysis of a model of the LDL degradation pathway, in: Gordon D. Plotkin (Ed.), Computational Methods in Systems Biology (CMSB'05), University of Edinburgh, 2005.
[35] Henrik Pilegaard, Flemming Nielson, Hanne Riis Nielson, Context dependent analysis of bioambients, in: Francesco Ranzato (Ed.), Proceedings of the 1st International Workshop on Emerging Applications of Abstract Interpretation, 2006.
[36] Corrado Priami, Paola Quaglia, Beta binders for biological interactions, Computational Methods in Systems Biology (CMSB'04), Lecture Notes in Bioinformatics, Springer, 2004.
[37] Corrado Priami, Aviv Regev, Ehud Shapiro, William Silvermann, Applications of a stochastic name-passing calculus to representation and simulation of molecular processes, Inform. Process. Lett. 80 (1) (2001) 25–31.
[38] Aviv Regev, Computational systems biology: a calculus for biomolecular knowledge, PhD Thesis, Tel Aviv University, 2003.
[39] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, Ehud Shapiro, BioAmbients: an abstraction for biological compartments, Theor. Comput. Sci. 325 (1) (2004) 141–167.
[40] Aviv Regev, William Silverman, Ehud Shapiro, Representation and simulation of biochemical processes using the π-calculus process algebra, in: Proceedings of Pacific Symposium on Biocomputing (PSB 2001), 2001, pp. 459–470.
[41] Olin Shivers, Control flow analysis in scheme, PLDI'88: Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation, ACM Press, New York, NY, USA, 1988, pp. 164–174.