Language Based Techniques for Systems Biology

Henrik Pilegaard

Kongens Lyngby 2007
IMM-PHD-2007-184

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Summary

Process calculus is the common denominator for a class of compact, idealised, domain-specific formalisms normally associated with the study of reactive concurrent systems within Computer Science. With the rise of the interaction-centred science of Systems Biology a number of bio-inspired process calculi have similarly been used for the study of bio-chemical reactive systems.

In this dissertation it is argued that techniques rooted in the theory and practice of programming languages, *language based techniques* if you will, constitute a strong basis for the investigation of models of biological systems as formalised in a process calculus. In particular it is argued that *Static Program Analysis* provides a useful approach to the study of qualitative properties of such models.

In support of this claim a number of static program analyses are developed for Regev's BioAmbients – a bio-inspired variant of Cardelli's Ambient Calculus that incorporates all features of Milner's π-calculus:

The property of spatial reachability, which is related to the function of cellular transport mechanisms, is addressed by two traditional Control Flow Analyses (CFAs). The simpler of the two, a mono-variant analysis (0CFA), is context insensitive, while the other, a poly-variant analysis (2CFA), is context-sensitive. These analyses compute safe approximations to the set of spatial configurations that are reachable according to a given model. This is useful in the qualitative study of cellular self-organisation and, e.g., the effects of receptor defects or drug delivery mechanisms.

The property of sequential realisability. which is closely related to the function of biochemical pathways, is addressed by a variant of traditional Data Flow Analysis (DFA). This so-called 'Pathway Analysis' computes safe approximations to the set of reaction sequences that is realisable according to given model. This

is useful in the qualitative study of the metabolic pathways that emerge from a group of connected biochemical agents.

Technically, these approaches are complementary, but the analyses all over-approximate the set of run-time enabled reactions. This is used in an iterative narrowing scheme that achieves considerable synergy between CFA and DFA, and dramatically improves the results of both.

The specified analyses are proved correct with respect to the semantics of Bio-Ambients, and their strength is illustrated by application to abstract models of biological phenomena:

One is a model of the LDL degradation pathway, where it is shown that the analyses are able to pinpoint the effects of certain genetic defects that are known to be associated to cardiovascular disease.

The other is a model of genetic transcription that relies only on the π calculus fragment of BioAmbients.

In both cases the analyses compute very precise estimates of the temporal structure of the underlying pathways; hence they are applicable across a family of widely used bio-ware languages that descend from Milner's Calculus of Communicating Systems.

The presented set of analyses constitutes a nice toolbox for the analysis of biological models. The individual tools range in complexity from low polynomial to exponential, while the precision scales similarly. Thus, the toolbox may provide useful information at all stages of a models lifetime, including *development*, where one is interested in frequent quick estimates, *verification*, and *prediction*, where one is willing to wait longer for more precise estimates.

Resumè

Process kalkule fungerer som fællesbetegnelse for en klasse af kompakte idelaiserede domæne specifikke formalismer, der normal associeres med studiet af reaktive systemer indendfor datalogien. I forbindelse med udviklingen af den interaktionsorienterede videnskab, der nu kaldes systembiologi, er et antal biologisk inspirerede process kalkuler ligeledes blevet brugt som basis for studiet af biokemiske reaktive systemer.

I denne afhandling argumenteres for at sprog-basserede teknikker, der har deres rod i det teoretiske og praktiske fundament for programmeringssprog, udgør et stærkt udgangspunkt for analysen af modeller af biologiske systemer, der er formaliserede i en process kalkule. Særligt argumenteres der for, at statisk program analyse er et nyttigt værktøj til studiet af de kvalitative egenskaber ved sådanne modeller.

For at understøtte denne påstand er der udviklet en række analyser til Regevs BioAmbients kalkule, der er en biologisk inspireret variant af Cardellis ambient kalkule, some desuden indeholder Milners π kalkule.

De rumlige egenskaber ved modeller, der er relevante for de cellulære transport mekanismer, angribes med to traditionelle kontrol flow analyser (CFA).Den simpleste af de to - en mono-variant analyse - er ikke sensitiv overfor kontekst, men den anden - en poly-variant analyse - er. De beregner begge sikre tilnærmelser til mængden af spatielle konfigurationer, som ifølge modellen kan nås fra en given starttilstand, Dette er nyttigt for det kvalitative studie af cellers selvorganisering og, for eksempel, effekten af defekte receptorer eller medicinale fremførings mekanismer.

De tidslige egenskaber ved modeller, der er relevante for biokemiske stier generelt, angribes med en variant af traditionel data flow analyse (DFA). Denne såkaldte 'pathway analyse' beregner sikre tilnærmelser til mængden af transitionssekvenser, som ifølge modellen kan realiseres af et givent system.

Teknisk set er disse tilgange komplæmentære, men begge overaproksimerer mængden af reaktioner som modellens dynamik tillader. Dette udnyttes i en iterativ tilgang, som opnår betydelig synergi mellem CFA og DFA og derved markant forbedrer resultatet af begge.

Det bevises at analyserne er korrekte med hensyn til sprogets (BioAmbients) formelle semantik og deres styrke illustreres ved brug på abstrakte modeller af biologiske fænomener:

Den ene modellerer den sekvens af biologiske operationer, der leder til optagelse og nedbrydelse af Low Density Lipo-protein i den den eukaryotiske celle. Det eftervises her, at analyserne er i stand til at udpege effekten af genetiske fejlkodninger, der vides at være forbundet med hjerte-/kar-sygdomme.

Den anden modellerer genetisk transkription på et meget abstrakt niveau, og kun ved brug af π kalkule delen af BioAmbients sproget. Det faktum, at analyserne i begge tilfælde er i stand til at uddrage meget præcise estimater af de underliggende stiers tidslige struktur, viser at de kan bruges på et spektrum af biologisk inspirede kalkuler der stammer fra Milners Calculus of Communicating Systems.

De udarbejdede analyser udgør en slagkraftig værktøjskasse til brug ved analyse af modeller af biologigiske systemer. De individualle værktøjer spænder kompleksitetsmæssigt fra lav polynomiel til eksponentiel kompleksitet, mens præcisionen skallerer tilsvarende. Dermed kan værktøjskassen levere brugbar information i alle faser af en models liv, lige fra udviklingsfasen, hvor man er interesseret i hurtige estimater relativt ofte, over verfikationsfasen, til driftsfasen, hvor man i højere grad er villig til at vente på præcise estimater.

Preface

This dissertation was prepared at the department of Informatics and Mathematical Modelling, Technical University of Denmark, in partial fulfillment of the requirements for acquiring the Ph.d. degree in engineering. The Ph.d. study has been carried out under the supervision of Professor Flemming Nielson and Professor Hanne Riis Nielson.

Acknowledgements. I am grateful to my supervisors, Flemming Nielson and Hanne Riis Nielson, for their continuous support and guidance. Furthermore I thank them, as well as the rest of the LBT group: Jörg Bauer, Han Gao, Christoffer Rosenkilde Nielsen, Sebastian Nanz, Christian W. Probst, Terkel Tolstrup, Ye Zhang, and Fan Yang, for providing an inspiring and motivating working environment.

A special thanks goes to Terkel Tolstrup for being a good source of inspiration and alternative viewpoints, as well as a good friend, both during our three years of sharing an office and after. I also thank Terkel Tolstrup, Christoffer Rosenkilde Nielsen, and Christian Probst for commenting on various fragments of this dissertation.

I thank Professor Mila Majster Cederbaum and Verena Wolf, Univesität Mannheim, for enthusiastically welcoming me into their group and making the arrangements for a visit that was, regrettably, cancelled for health related reason.

Finally, I am in debt to family and friends, especially Anette and Leonora – the gems of my life, for their patience and support, without which I could not have finished this dissertation.

Lyngby, July 2007

Henrik Pilegaard

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"How much broader our view of life would be if we could study it through reducing glasses."

— Louis Bolk

Throughout time the concept of life has been one of the most fundamental mysteries of every human society. And, even though a great number of highly motivated people, some of whom remarkably clever, have spend entire lifetimes pondering the subject, it has remained elusive to us.

This might be about to change. The technological revolution of the last decades has drastically accelerated the traditional reductionist bottom-up approach to Life Sciences. As a result the bottlenecks of Life Sciences now lie beyond the data acquisition phase, where massive amounts of data are now easily collected by cell-wide measurements of transcriptomes, proteomes, metabolomes, etc. [vSvdH04].

The first main insight spawned by these post-genomics approaches is that no observable property of the cell emerges as a simple function of a single gene or protein. Rather it has become evident that individual biochemical entities merely enact roles within one or more elaborate clusters of entities that interact cooperatively in a systematic manner in order to cause the observable behaviour. Thus, the main challenge now is to identify and describe *systems*, rather than individual molecules [vMJ+07].

This challenge has been accepted by the emerging science of *Systems Biology*, which aims squarely for a holistic understanding of life. The basic tenet of Systems Biology is that in order to understand life we must understand both

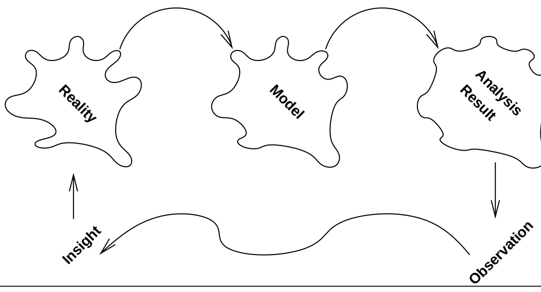


Figure 1.1: The goal of modelling based approaches.

the qualitative and the quantitative aspects of the underlying systems [Kit02].

For a fraction of Systems Biology, known as *Computational Biology* – the area where this dissertation contributes – the goal is the scenario depicted in Fig. 1.1, where *in vivo/in vitro* experiments, such as those involved in *drug development*, are carried out *in silico*. Basically, we hope to compute important properties of the connectivity and self-organisation of systems from *mathematical models* that incorporate our knowledge about the individual constituents and the communication and self-organisation mechanisms that are available to them [van05].

An open issue in the modelling approach of Computational Biology is that of representation. In recent years, however, it has become increasingly clear that *reactive systems*, which have been studied intensively within the Computer Science area of Concurrency Theory for 25 years, are in many ways similar to the systems of biology. In both cases the systems of interest are composed from collections of independent, yet somehow connected, agents, either primitive or themselves systemic, the interactions of which give rise to the global behaviour of the systems [RS02].

Thus it is hardly surprising that Process Calculi, a class of elegant algebraic domain-specific modelling languages that has been tremendously successful in the study of reactive systems, is now also widely applied, and studied, in the context of Computational Biology [Reg03, Car05b, DL04, PQ04].

These languages are intensional rather than extensional; hence process calculus models are *operational* in the manner of computer programs — a fact that might become a crucial factor when making the step from *drug discovery* to *drug design* [Car04]. Process calculi are also *compositional* in the sense that the properties of a model emerges, in a well-understood manner, from the properties of the immediate constituents and the method of composition [Mil99] — a feature that is very desirable both from a modelling and analysis perspective. Finally, while it is common for process calculi to focus on qualitative aspects, it is well-understood how to integrate quantitative information [Hil96]. This can even be done in a totally orthogonal manner, which enables a convenient separation of concerns [Her02].

However, process calculi are also Turing complete, i.e., powerful to the point where many, not to say most, interesting properties of models cannot be computed within finite time and memory [Tur36] — a feature that is hardly acceptable to a research field like Systems Biology, which relies heavily on fully automated analysis.

In order to circumvent this issue, computational biologists often defer to simulation in favour of formal verification, e.g., [Reg03, PC04]. In a sense this is a fairly direct approach to the investigation of models, as it relies on execution of the underlying formal semantics. Usually, however, the method relies on sampling, rather than exhaustive investigation, and hence it provides probabilities rather than formal assurances.

Another option is to go the last mile and perform *exhaustive simulation.* This is the approach of finite state *Model Checking* [CGP00], which often works well for state spaces of moderate size, but, due to the so-called state space explosion, becomes intractable for large state spaces and undecidable for infinite ones. Thus, in the presence of complicated models this approach often fails to provide formal assurances.

Abstraction is the key to such problems. This was realised decades ago within the Computer Science area of Compiler Construction. Traditionally, *optimising compilers* rely on the fully automatic, but approximative, approach of *Static Program Analysis* in order to provide the assurances required in order to safely optimise a program into another, more efficient program, while preserving the desired extensional behaviour [NNH99]. The basic tenet of Static Program Analysis is that a loss in precision often makes a property decidable. Thus, the fundamental challenge of the approach is to achieve an appropriate balance between efficiency and precision.

Main Thesis. This leads to the *main thesis* of this dissertation:

> The approximative approach of Static Program Analysis can be used to automatically decide biologically relevant properties of process calculus expressions that model biological systems.

In principle, any property that can be used to pinpoint errors during model development or predict the consequences of perturbations qualifies as *interesting.*

For the purpose of this dissertation, however, the focus will be on the *qualitative* aspects of the *spatial* and *temporal structure* of models.

1.1 Background

In order to investigate this thesis the dissertation combines the following three elements:

An Object Domain. The object of interest to this dissertation is the eukaryotic cell – the smallest living component of mammalian organisms – as well as its various subsystems. This constitutes a homeostatic system with two main features:

The cell is an amazingly flexible *reactive system* that quickly and precisely responds to a huge array of environmental stimuli. This ability is intimately connected with a highly evolved information processing capability. The underlying processing plant is implemented by a set of tightly connected regulatory networks.

The cell is a robust *self-organising system* that maintains an almost infallible spatial separation between the components of external and, various, internal environments. The underlying compartment system constitutes an elaborate network of bio-membranes that define compartment boundaries. The essential differences between the fluids of the various compartments are maintained by a collection of transport networks.

These aspects emerge as high-level behaviours from highly connected systems of interacting bio-chemical agents. Each of the networks comprises a number of metabolic pathways that constantly produce, modify, and degrade bio-chemical agents in a selective manner.

A Modelling Formalism. Process calculi have established themselves as tools of choice for the modelling and analysis of concurrent and reactive systems. The use of process algebraic formalisms outside of traditional computing contexts is a growing trend and, in particular, the emerging area of *Systems Biology* has received a lot of attention from the process algebra community.

This dissertation commits to this movement by using a variant of the BioAmbients calculus [RPS$^+$04] as the basis for modelling and analysing subcellular biological systems. The BioAmbients calculus is a descendant of the CCS [Mil80] family that extends both the π calculus [Mil99] and the ambient calculus [CG00]. It retains a notion of *spatial boundaries*, called ambients, from the ambient calculus, which allows for models that preserve the spatial structure of the bio-domain. Furthermore, a notion of *interaction capabilities*, in the style of the synchronous π calculus, allows the modelling of biological reactions between sub-systems.

Thus, the calculus is equally well suited for the modelling of regulatory networks, cellular transport networks, and the underlying metabolic pathways.

A Formal Approach to Model Analysis. The notion of *approximation* is pervasive in the application of mathematics in the analysis of real-world phenomena. If a model is too strong, seemingly simple properties become intractable, and, if it is too weak, the computable properties are of no interest. Thus, engineers and scientists alike habitually walk the edge of Ockham's razor and introduce simplifying assumptions in order to strike sensible compromises between the accuracy of estimates and the tractability of computation.

In the context of computer programs, however, such simplifying assumptions are of no help as programs are normally intended to do exactly what the instructions describe. The solution proposed by Static Program Analysis [NNH99] is to incorporate the approximation into the analysis techniques rather than the models. The result is a set of techniques that systematically perform approximations in order to calculate some aspect of the behaviour of a model by inspection, rather than execution, of the program code.

For this approach to work analyses are usually required to be 1) *correct* with respect to the formal semantics, such that verification of a property implies assurance, 2) *exhaustive* in order not to restrict the programmer, 3) *fully automatic*, in order to pose minimal requirements on the programmer, and 4) *efficient*, in order to handle the complexity of more realistic models.

It is exactly this set of required features that makes the approach attractive in the context of Computational Biology also.

1.2 Contributions

In this context this dissertation contributes a number of static analyses in support of the main thesis. It is proved that the specified analyses are both exhaustive and correct with respect to BioAmbients models. Furthermore, it is shown that the analyses admit the computation of best analysis results and thus are implementable; all results presented in this dissertation have been automatically computed by working prototypes.

Mono-variant Control Flow Analysis. In accordance with the ordinary approach of Flow Logic based program analysis, the first analysis is a *context insensitive* or *mono-variant Control Flow Analysis* (*0CFA* [Shi88]) for the BioAmbients language. The analysis aims to safely over-approximate the set of spatial configurations that are reachable from a given model. In order to do so it keeps track of the contents of ambients and their abilities for participating in various interactions.

Thus, the focus of the approximation is on the spatial hierarchy established by the nesting of ambient boundaries. The analysis takes the perspective that a nesting hierarchy is a tree, but only approximates the shape of this tree by a binary relation $\mathcal{I}$ between ambients and sub-ambients or capabilities, i.e., $(a, b) \in \mathcal{I}$ means that the ambient or capability b occurs inside the ambient a.

Analysis results are considered acceptable if the associated relations, within the limited precision of the relational representation, a) faithfully represent the initial configuration, and b) are closed with respect to conditions that mimic the semantics. In this respect the approach is similar to Abstract Interpretation [CC77].

Parts of this work were previously presented in [NNPdR07].

Poly-variant Control Flow Analysis. The second analysis is a *context sensitive* or *poly-variant Control Flow Analysis* (*2CFA* [Shi88]) for the BioAmbients language. The aims and means of the analysis are similar to those of the aforementioned 0CFA.

However, where the 0CFA approximates trees in terms of a binary relation the 2CFA does it in terms of a quaternary relation, where $(a, b, c, d) \in \mathcal{I}$ means that d is inside c, c is inside b, and b is inside a — or d is inside c *in the context of* a, b.

This context information makes the analysis *context sensitive*, i.e., able to differentiate between (sub-)configurations that are possible in different settings. Not only does this, in itself, yield a more precise representation, but it also allows for closure conditions that mimic the semantics much more closely.

Parts of this work were previously presented in [PNN06b].

Pathway Analysis. The CFA analyses safely approximate the set of spatial configurations that may arise at run-time. In contrast they produce no information about the sequential order of the transitions that lead to the recorded configurations.

This motivates the development of the third analysis: a flow sensitive *Pathway Analysis* that aims to safely over-approximate the set of reaction sequences that are realisable by a given model. In order to do so, it focuses on the notion of *exposed prefixes*, i.e., the capability prefixes that *might* participate in the next reaction.

Technically, this analysis resorts to techniques normally associated with classical Data Flow Analysis [KU77]. The analysis takes the perspective that a state, i.e., a process, is an *extended multiset* of exposed prefixes. It then computes a (partially) Deterministic Finite Automaton, that safely approximates the set of realisable transition sequences, by tracking how these multisets change when reactions occur.

Parts of the work presented here have also been submitted as part of [PNN07].

Iterative Analysis. At first sight the two types of analyses are quite different: The CFA analyses approximate the spatial structure of models, and, in contrast, the Pathway Analysis approximates the temporal structure.

These differences, however, are not as profound as they might appear. Clearly, the analyses are concerned with different aspects of the same thing, namely the run-time behaviour of BioAmbients processes, and, regardless of the analysis approach, this is closely related to the realisable reactions, i.e., the set of reactions that might occur. Indeed, both the CFAs and the Pathway Analysis rely on preliminary safe estimates of this set in order to improve precision, and both of them compute new, safe – but smaller – estimates as part of the analysis results.

Due to their inherent differences, the two types of analysis often produce different estimates of the realisable reactions. This is used in an iterative narrowing scheme that alternates the application of a CFA and the Pathway Analysis until the estimate of realisable reactions reaches a fixed point.

While this is remarkably simple, it improves the precision of both CFAs and Pathway Analysis dramatically.

This work has not been presented previously.

Case Studies. The analyses are evaluated in the context of two case studies that address very different aspects of the eukaryotic cell.

The first case study deals with the internalisation of Low Density Lipo-protein, and is thereby related to cellular self-organisation. The associated model is fairly concrete and emphasises the spatial features as well as the receptor dynamics of the LDL degradation pathway. The modelled system is related to known cardiovascular diseases; this plays a key role in the investigation of the model, which is analysed with the full battery of static analyses.

The second case study deals with genetic transcription, and is thus related to cellular information processing. The associated model is very abstract and relies only on the π calculus fragment of BioAmbients. Due to the omission of spatial properties the model is primarily investigated with the Pathway Analysis.

The former model and parts of the associated analytic investigation was previously treated in [PNN05]. The latter model is new.

1.3 Preliminary Conclusion

Can Static Analysis decide interesting properties of biological systems?

The conclusion, based on the work and results to be presented in the following, is that yes, it can. The established battery of analyses, which ranges in complexity from (low) polynomial to exponential, is able to precisely, and with increasing accuracy, pinpoint the most essential aspects of the studied case models.

In particular, the analyses succeed in pinpointing the effects of certain genetic defects, known to be associated with cardiovascular disease, from the model of the LDL degradation pathway. Furthermore, the analyses are able to extract very precise estimates of the metabolic pathways that emerge from both models; a fact that indicates that the analyses are applicable across an entire family of widely used bio-ware languages that descend from Milner's CCS.

Thus the set of analyses presented in this dissertation constitutes a strong tool that can both *support the development of models* and *automate significant parts of their post-modelling analysis*. The former point in particular is supported by the, subjective, practical modelling experiences of the author.

1.4 Dissertation Outline

The present dissertation contains ten chapters, of which this introductory chapter is the first. The remaining nine chapters are organised into three parts:

Part I is mainly concerned with background material.

Chapter 2 examines the main features of the eukaryotic cell – including the two phenomena that are modelled and analysed in later chapters: the process of genetic transcription and the LDL degradation pathway.

Chapter 3 explains the central notions of process calculi in a biological context, and, in particular, the 'biological compartment as computational ambient' abstraction of the BioAmbients modelling language. Furthermore, the chapter presents the two case models to be analysed in later chapters.

Chapter 4 introduces some well established approaches to static analysis — both the classical Monotone Frameworks approach to Data Flow Analysis and the more recent Flow Logic approach to Control Flow Analysis.

Part II is concerned with the use of Control Flow Analysis (CFA) techniques for approximating spatial properties of BioAmbients models.

Chapter 5 formalises some of the subtler concepts of the BioAmbients calculus and defines a notion of well-formed programs.

Chapter 6 specifies a classic context-independent Control Flow Analysis (0CFA), which safely over-approximates the set of reachable spatial configurations of any well-formed BioAmbients program, and submits the model of the LDL degradation pathway to analysis.

Chapter 7 presents a context-dependent Control Flow Analysis (2CFA), which constitutes an improvement of the 0CFA, and submits the model of the LDL degradation pathway to analysis.

Part III is concerned with the use of Data Flow Analysis (DFA) techniques for approximating temporal properties of BioAmbients models.

Chapter 8 introduces the Pathway Analysis, a DFA that safely over-approximates the set of realisable causal sequences of any well-formed BioAmbients program, and submits both of the case models to analysis.

Chapter 9 shows how the developed CFAs and DFA can be combined into a single iterative analysis and submits both of the case models to analysis.

Finally, *Chapter 10* summarises, concludes, and presents the perspectives for further work.

Part I

Setting the Scene

Chapter 2

The Eukaryotic Cell

"It is astonishing to think that this remarkable piece of machinery, which possesses the ultimate capacity to construct every living thing that ever existed on Earth, from giant redwood to the human brain, can construct all its own components in a matter of minutes and weighs less than 10^{-16} grams. It is of the order of several thousand million million times smaller than the smallest piece of functional machinery ever constructed by man."

— Michael Denton

In this chapter we review the main features of the *eukaryotic cell*. The goal is to establish a rudimentary understanding of the biological domain. In order to fix a universe of discourse for the later technical developments we shall primarily be interested in the *qualitative* aspects of cellular information processes and cellular transport. The material covered can be found in any good textbook on the molecular biology of the cell, such as [AJL$^+$02] or [LBZ$^+$99], and contains no original contributions.

We start, in Section 2.1, with a brief note on the nature of biochemical interactions. Then we give an overview of cellular information processing and introduce the underlying bio-chemical polymer plant. This material will help to appreciate the basic molecule-as-reactive-process abstraction of later chapters. Finally, in Section 2.2, we describe the organisational hierarchy of eukaryotic cells and discuss the underlying self-organising compartment technology. This material will help to appreciate the more advanced compartment-as-computational-ambient abstraction of later chapters.

2.1 Cellular Information Processes

The cell is a reactive system. This ability is intimately connected with a highly evolved information processing capability. The underlying processing plant is implemented by a highly connected system of interacting *bio-polymers* — the class of *polymers* that are produced by living organisms.

Molecular Complementarity. The notion of *molecular complementarity* is central in all chemical interactions. Like most relationships, the bonds formed by atomic and molecular entities are based on the mutual satisfaction of interests.

Some of the formed relationships are very strong and may form stable *molecules* from otherwise separate entities. This is the case for *covalent bonds* caused by the sharing of electrons between atoms of *complementary valence.*

Other relationships are weaker and may form stable or less stable molecular *complexes. Ionic bonds* are caused by *electrostatic forces* between atoms of *complementary charge*, i.e., one is *electronegative* and the other *electropositive. Hydrogen bonds* are caused by (weaker) electrostatic forces between an electronegative atom and an (eletropositive)) hydrogen atom bound in a dipolar constellation to another electronegative atom. Finally, *van der Waals interaction* is a weak unspecific attractive force between atoms in close proximity.

Water, the solvent of the cell, is an example of a *dipole*; it exhibits a small positive charge near the hydrogen atoms and a small negative one near the oxygen. The *water solubility* of molecules is determined by their *electrochemical properties*: *Charged molecules* are generally soluble, whereas the solubility of *uncharged molecules* is determined by their ability to form hydrogen bonds with water molecules. Hydrophobic (water insoluble) molecules tend to associate tightly when submerged in water. This is called the *hydrophobic effect.*

2.1.1 Bio-molecular Agents and their Interactions

For the small molecules typically found in ordinary solution chemistry it is easy to predict the effects of these various forces. This is different, however, for bio-molecular agents that are mostly long *polymeric chains* composed of smaller *monomeric units* with individual electrochemical properties. The complicated structure of these entities results in a very involved notion of molecular complementarity that induces a high degree of *binding specificity.* This is one of the features that most distinguish *biochemistry* from typical *solution chemistry.*

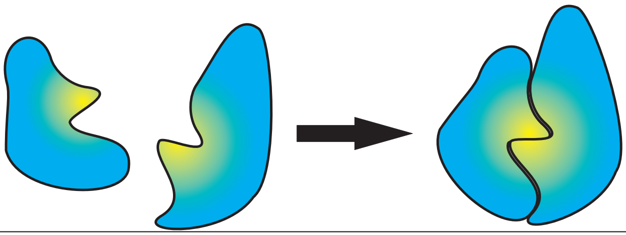


Figure 2.1: Protein-protein interaction.

Three types of bio-polymers play a central role in cellular information processing:

Proteins. The *proteins* are the main actors of the cellular *metabolism*, i.e., the set of chemical reactions that occur in the living cell. They are sequences of *amino acids*, i.e., simple acids consisting of a *residue*, an *amino group*, and a *carboxylate group*. About 20 different amino acids occur in living organisms.

The properties of any protein are determined by the sequence of amino acids from which it is composed. This sequence constitutes the *primary structure* of the protein.

In turn, the properties of each individual amino acid are determined by its acid residue, which may be *non-polar*, *basic*, *acid*, or *uncharged polar*. The diversity of these properties allows non-trivial interaction patterns to emerge within the primary structure. In particular, *hydrogen bonds* may form between certain residues and cause stable localised foldings, such as *α-helices*, *β-sheets*, and *turns*. The resulting spatial arrangements constitute the *secondary structure* of the protein.

The properties of the resulting chain of secondary structure elements are largely determined by the properties of the individual substructures. Some sections may be *hydrophilic* (water soluble), and others hydrophobic. When submerged in water, proteins seek the most stable conformation and invariably fold to hide hydrophobic sections and expose the hydrophilic ones. This is an example of the hydrophobic effect. Internal formation of strong or, more commonly, weak bonds further stabilises the folded structures. The resulting 3-dimensional conformations constitute the *tertiary structure* of the protein.

The properties of a protein that has folded into a stable conformation are determined by the resulting *solvent-accessible surface*. The physical and electrochemical contours of the surface are characteristic for the protein. Two folded polymers may exhibit surface areas of (nearly) complementary contour and these *binding sites* then allow them to interact biochemically. If the binding sites match well they allow the formation of many stabilising bonds. The molecules then have a high *affinity* for one another and may form stable structures called *complexes* or *coordination compounds*. The number and organisation of subunits in such a compound constitutes the *quaternary structure*. If the interaction is a brief *reaction* and the protein is a *catalyst* of change in the other molecule we call it an *enzyme*. In contrast, if the protein is changed by the interaction we call it a *substrate*.

DNA. *Deoxyribonucleic Acids* are the main carriers of cellular hereditary information. They are sequences of *nucleotides*, i.e., monomeric units consisting of a base, a sugar, and one or more phosphate groups. In *DNA* the sugar is always a *deoxyribose* and the bases are *Adenine*, *Guanine*, *Cytosine*, and *Thymine*.

Every cell has a repository of hereditary information stored in DNA. The smallest unit of heredity is the *gene*. The collection of all genes present in a cell is the *genome* and, under normal circumstances, this is an extremely stable entity.

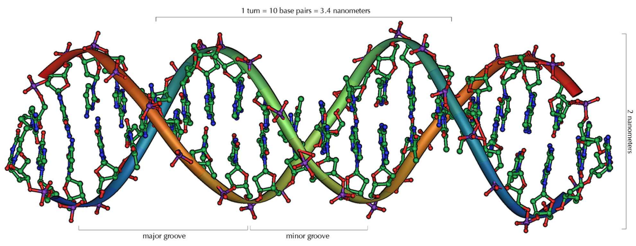


Figure 2.2: DNA double helix [Jon07].

This stability owes to the structure of DNA. The polymer sequence is formed as the sugar phosphates are linked up sequentially to form the *backbone*. From this backbone the bases protrude as a sequence of stubs. The bases are prone to the formation of *hydrogen bonds* between pairs (A-T and C-G). Thus, *complementary strands* can *base-pair* and link up in *anti-parallel* to form a very stable *duplex* shaped like a *double helix*. Genetic information is invariably stored in this form.

It is usual to represent DNA as strings over the corresponding four letter alphabet: A,C,G,T. The sequence is directional; one end is denoted by $5'$ and the other by $3'$ – denotations that reveal the positions of recognisable terminators on the sugar-ring of the first and last molecule synthesised, respectively. Thus, DNA and RNA are invariably synthesised in the direction of $5' \rightarrow 3'$, which is called *downstream*. The opposite direction is *upstream*.

RNA. *Ribonucleic Acids* are the main facilitators of interaction between protein and DNA/RNA. They are nucleic acids where the sugar is a ribose and the bases are adenine, cytosine, guanine, and *Uracil*.

Structurally, RNA is very similar to DNA. In contrast to DNA, however, it mostly occurs as comparatively short single-stranded chains of nucleotides and, due to the extra hydroxyl group of ribose, it is more prone to hydrolysis. Consequently RNA is relatively unstable and tends to fold into more stable (3-dimensional) conformations in order to stabilise. Folded RNA molecules often forms mixed ribonucleoprotein (RNP) complexes with proteins.

Thus, RNA strands can act both as information carriers (*coding RNA*) and functional units (*non-coding RNA*):

Coding RNA is synonymous with *messenger RNA* (mRNA). This type of molecules carry transcripts of genes that encode proteins from the genome to the *ribosomes*, where the proteins are produced in accordance with the transcribed information.

The major types of *non-coding RNA* also play important roles in the transfer of information from DNA to proteins. *Ribosomal RNA* (rRNA) molecules constitute the main part of the *ribosomes*, the enzymes that read mRNA in order to produce proteins. Two *ribosomal ribonucleoproteins* (rRNPs), known as the *large* and the *small subunit*, respectively, form a functional ribosome. *Transfer RNA* (tRNA) molecules are the adaptors that select and hold individual amino acids in place for the ribosomal processing.

Other types of non-coding RNA serve in various regulatory capacities throughout the information transfer process.

2.1.2 The Central Dogma

The *Central Dogma of Molecular Biology*, first pronounced by Crick in 1958 [Cri58], states that the molecular flow of information is from DNA via *transcription* to RNA and from RNA via *translation* to protein(s). As shown in Figure 2.3 there are known exceptions, but these are associated with abnormal conditions.

CASE: Transcription of Genes. Each gene encodes either a set of *protein isoforms* or a non-coding RNA string. The first step in actually producing these entities is the *transcription* of the corresponding DNA into RNA.

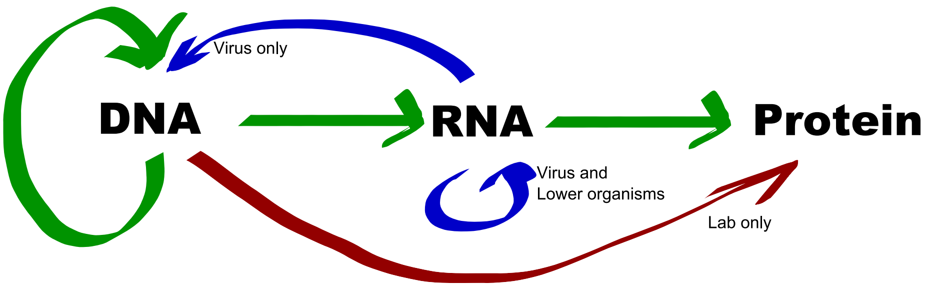


Figure 2.3: Flow of information in biological systems.

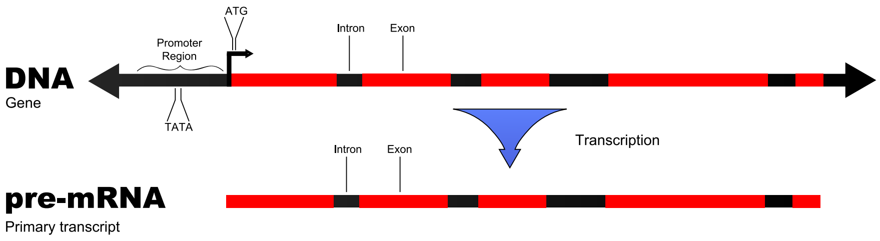


Figure 2.4: The transcription of genes [Jon07].

The stretch of DNA that is the gene consists of two regions. The *coding region* is what describes the actual product(s) and upstream from that is the *promoter region*.

A number of enzymes are involved in the transcription process, which has three phases: During *initiation* an *RNA polymerase* attaches at the promoter and *melts* the DNA locally. During *elongation* the polymerase synthesises a *primary transcript* RNA from the *sense strand* (the one that codes the gene) by chaining of *nucleoside tri-phosphates* (NTPs). Finally, during *termination* the *nascent* (growing) RNA strand and the polymerase are released from the DNA.

Transcription is heavily regulated. A number of *regulatory regions*, located in the upstream or downstream area of the promoter, accommodate the binding of *transcription factors*. These affect the affinity of the promoter for RNA polymerase and may be *activators* or *repressors*, depending on their influence.

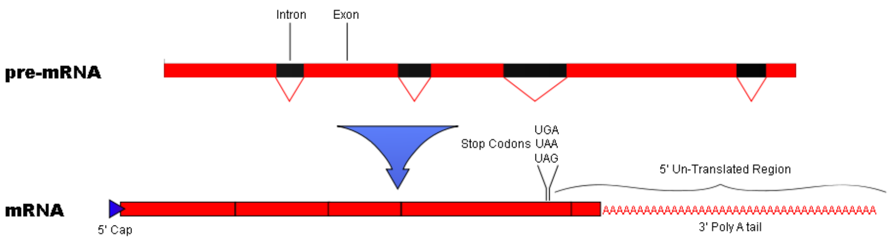


Figure 2.5: Post-transcriptional processing of pre-mRNA into mRNA.

The gene usually contains both coding regions (*exons*) and non-coding regions (*introns*). The introns are removed post-transcriptionally from the primary transcript, called *precursor mRNA* (pre-mRNA), by a process called *splicing*. Regulated variations in this process allows a single gene to code a family of mRNA.

Translation of RNA. *Translation* begins when the small ribosomal subunit assembles on the mRNA and seeks out a *start codon*. Once the start codon is found the large subunit assembles and translation commences by stepwise *elongation*. When a *stop codon* is encountered the subunits disassemble and translation terminates.

Each elongation step consumes a tRNA molecule. At one end the tRNA has a three-nucleotide sequence (*anti-codon*) that can base-pair to a matching three-nucleotide fragment (*codon*) of mRNA. At the other end they have a binding domain that matches one of the twenty available amino acid monomers. This implicitly defines the *genetic code*.

Proteins are folded and subjected to various enzyme-induced modifications as they emerge from the ribosome during translation.

2.2 Cellular Organisation

The cell is highly organised and robust due to an amazing capacity for self-organisation. The underlying compartment system constitutes an elaborate network of *bio-membranes* that define *compartment boundaries* and maintain the essential differences between the inside fluid and the outside fluid.

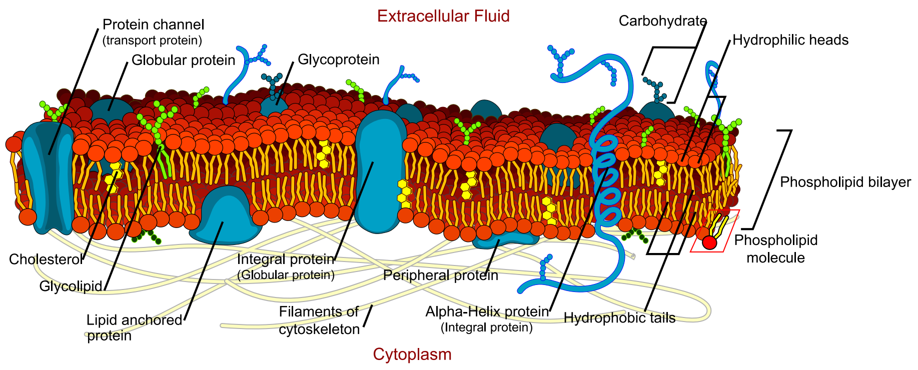


Figure 2.6: Structure of the cell membrane [Vil07b].

2.2.1 Lipid Bi-layer Membranes

Bio-membranes are bi-layered sheets. As shown in Fig. 2.6 they are mainly composed of *amphiphatic lipid molecules* and *membrane proteins* that perform free lateral diffusion on the surface. As the two monolayers face different biochemical environments their composition is different; hence the bi-layers are oriented.

Phospholipids. *Phospholipids* are the main constituents of bio-membranes. They consist of a hydrophilic *head* and two hydrophobic *tails* made from *fatty acids*. In order to hide their hydrophobic tails from polar molecules they spontaneously form self-healing spheres of *bi-molecular sheets* when submerged in water.

Membrane lipids perform rapid *lateral diffusion* within the same mono-layer, or *leaflet*. Thus, the bi-layer acts as a well-stirred 2-dimensional fluid. *Transverse diffusion*, from one mono-layer to the other, is rare because the hydrophilic head would have to penetrate the hydrophobic interior of the bi-layer.

Cholesterol. *Cholesterol* molecules usually intersperse the membrane lipids. Their presence increases the rigidity of the membrane and also lowers the freezing temperature by reducing the interaction between molecules. Careful regulation of their prevalence allows a fine-grained control over the *small molecule permeability* of membranes.

Membrane Proteins. Membranes are not just passive barriers. They provide *selective permeability* and are generally sites of considerable metabolic activity. The mediators of these functions are the *membrane proteins*.

Transmembrane proteins consist of two *functional domains* that protrude on either side of the membrane and are separated by a *beta barrel* (a stable formation comprising multiple β-sheets) or a number of regular α-helices that extend through the membranal bi-layer.

Anchored membrane proteins are attached only to one leaflet of a membrane. They are covalently bound to either a fatty acid or a phospholipid that is able to embed in the leaflet itself, thereby acting as an anchor.

Peripheral membrane proteins are associated to membrane surfaces by non-covalent interactions with membrane proteins or lipids. Such proteins usually adhere only temporarily to the membrane, and often in a regulatory capacity.

Like the membrane lipids, the proteins move by lateral diffusion. As they are richer in structure, however, they move considerably slower and never perform transverse diffusion.

2.2.2 Cellular Organelles

The eukaryotic cell, which is shown in Fig. 2.7, is not a monolithic structure. Rather it has elaborate internal structure in the form of membrane-bounded *organelles*. The membrane of each organelle maintains a *local environment* that is optimal for a set of highly specialised functions.

The most prominent compartment is the *cell*. It is bounded by the *plasma membrane*, which separates the *cytoplasm*, where the organelles float in *cytosol*, from the *exoplasm*. The outermost layer of the plasma membrane is the *exoplasmic leaflet*, and the innermost layer the *cytosolic leaflet*. The cytosol is an important site of metabolic activity. In particular this is where ribosomes translate mRNA into protein.

The *nucleus* is where the genome is maintained and transcribed into RNA. It is shielded by the *nuclear envelope* — a double-membrane penetrated by *nuclear pores* that facilitate the extremely selective exchange of materials between the *nucleoplasm* and the cytosol.

The *endoplasmic reticulum (ER)* has two parts: The *Rough ER* is involved in the folding and stabilisation of proteins for the plasma membrane or secretion. It is rough because the surface is studded with protein manufacturing ribosomes that attach as soon as the nascent proteins prove to be destined for the membrane or secretion. The *smooth ER* extends the rough ER and is involved in the synthesis of fatty acids and phospholipids for the membranes.

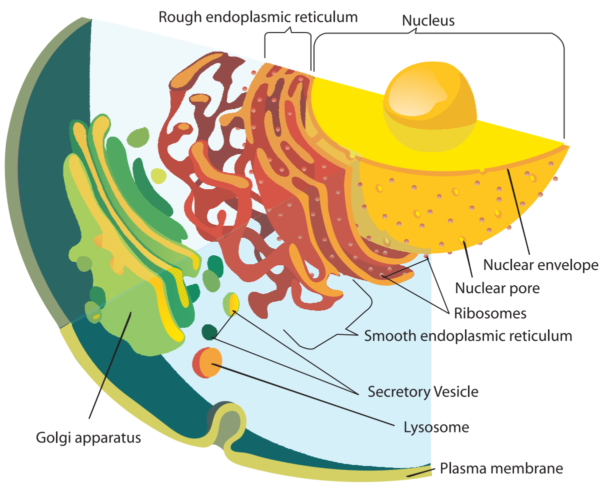


Figure 2.7: Structure of eukaryotic cells [Vil07a].

The *Golgi complex* sorts and processes membrane-bound and secretory proteins, and attaches *molecular labels* according to their transport destinations.

The *lysosomes* are the digestive units of the cell. They utilise enzymes to break down macromolecules and also act as a waste disposal system.

The *mitochondria* are responsible for energy production by aerobic respiration.

The *peroxisomes* are responsible for selective enzymatic oxidation of proteins and fatty acids.

Finally, *vesicles* are small, short lived, membrane-enclosed transport units that transfer molecules between different compartments. Vesicles form as bubbles that bud off the membrane of an existing compartment.

2.2.3 Cellular Transport

The most important function of the compartment membranes is to ensure the stability and organisation of the cellular environment. In particular it is important that the cellular environment is:

- electrochemically stable,

- biochemically well-organised,
- and effectively separated from the outside environment.

These properties are ensured by a number of transport mechanisms that, collectively, deal with all of these aspects.

Small Molecule Transport. The electrochemical stability of compartments is primarily maintained by the selective permeability of small molecules, which is implemented by a large class of transmembrane proteins.

Passive transport amounts to diffusion. Some gases may pass the membrane by *simple diffusion.* Slightly larger molecules, i.e., various ions and even water, pass by *facilitated diffusion* through *molecular channels.*

Active transport requires a source of energy. *Molecular pumps* hydrolyse ATP, while *molecular transporters* use the power of electrochemical gradients.

Large Molecule Transport. The biochemical organisation of the cell is maintained by the selective transport of large molecules, e.g., proteins, from production site to deployment site, The central mechanism is that of *protein targeting.* Every translated protein has one or more *signal sequences*, identifying the appropriate deployment site, embedded in its amino acid chain. Each such sequence is recognised by the transport machinery associated with the corresponding destination.

Non-secretory proteins are produced in the cytosol and subsequently transported to the lumen or membrane of an organelle. *Nuclear Localisation Signals* (*NLSs*), for example, are recognised by *Nuclear Pore Complexes* (*NPCs*) that facilitate transport from cytosol to nucleoplasm. Other signals direct proteins to the lumens or membranes of peroxisomes, or to the membranes or sub-compartments of mitochondria.

RNA molecules are produced in the nucleoplasm and, if appropriate, subsequently transported to the cytosol. In order to be recognised and transported by the NPCs they must form ribonucleoprotein complexes with proteins that exhibit *Nuclear Export Signals*(*NESs*).

The Secretory Pathway. *Secretory proteins* are meant for deployment in, or on membranes in contact with, exoplasmic solutions. Such solutions are rich in entities that contain important metabolites but are potentially harmful. Thus, many secretory proteins are *hydrolases*, i.e., enzymes that break down organic compounds, and cannot be allowed to roam freely inside the cell.

Signal recognition particles recognise secretory proteins already during translation, and immediately associate them with the rough ER surface. Here they are injected directly into either the membrane or the lumen of the rough ER by *co-translational translocation.*

Once folded into the proper conformation within the ER the proteins are packaged into *anterograde vesicles* and moved forward to the Golgi complex. Here they are matured and sorted into vesicles according to their final destination, which might be either the lysosome or the cell surface. Facilitating proteins are continuously shipped back to the ER in *retrograde vesicles.*

Finally, once fully matured, the secretory proteins leave the Golgi complex in vesicles. Some go to the plasma membrane, where they are secreted into the exoplasm by *exocytosis.* Acid hydrolases, on the other hand, are deployed to the lysosome, where they are used to break down organic compounds.

In the latter case the vesicle is coated with a double protein coat. The actual formation of the vesicle happens due to a coat of *Clathrin particles*, but underneath this coat there is another, comprised of *adopter protein complexes* (*APs*).

The Endocytic Pathway. The mechanism also facilitates *receptor mediated endocytosis*, which is used by the cell to selectively subsume particles from the exoplasm. The process is facilitated by specialised *transmembranal receptor proteins* whose exoplasmic domains are able to ligate specific particles.

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.*

The diffusing receptors tend to associate with clathrin coated pits because their *intra-cellular* domain binds to complementary *adaptin* (AP) molecules exposed by the clathrin coat. Such associated receptors and the particles 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 subsumed particles are still ligated by the receptor proteins. This changes, however, when the early endosome merges with a *late endosome.* The acidic environment in this compartment makes the receptors separate from the ligated particles.

From the late endosomes the receptor proteins are recycled to the plasma membrane. The internalised particles, however, are transferred by vesicles to *lyso-*

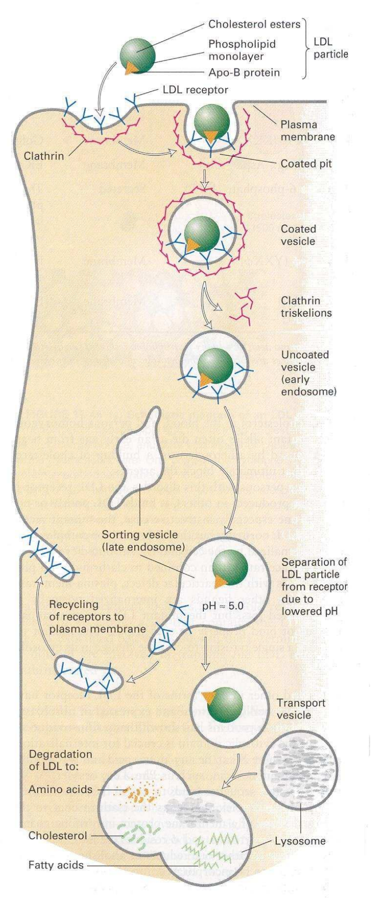


Figure 2.8: LDL degradation pathway. Copyright 2004 from Molecular Cell Biology by Lodish et al [LBZ$^+$99]. Reproduced by permission of W.H. Freeman and Company/Worth Publishers.

somes where they are broken down into useful metabolites,

CASE: The LDL Degradation Pathway. The best known example of this process is the *LDL degradation pathway* shown in Fig. 2.8. This is one mechanism, by which the cell acquires the *cholesterol* required for membrane synthesis. Transmembranal *LDL receptors* ligate LDL when the exoplasmic domain encounters the *ApoB* binding site exposed by LDL particles. The cholesterol is released when the tightly packed *cholesteryl esters* of the LDL particles are hydrolysed in the lysosome [AJL$^+$02, LBZ$^+$99].

Related Diseases. It happens that the gene encoding the transmembranal LDL receptor proteins somehow mutates. Sometimes these mutations are benign, and they cause no particular problems. It may also happen, however, that a mutation affects the coding of either the exoplasmic or the cytosolic binding domain in an adverse way. Either case leads to transcription of transmembranal receptor proteins that exhibit reduced affinity between the affected binding site and the corresponding binding sites of the ordinary ligands.

When such a defect affects the extra-cellular part of the receptor, its ability to bind LDL particles is reduced. In contrast, when the defect affects the intracellular part of the receptor protein, it can bind but not internalise LDL particles. Both cases lead to abnormally high blood levels of LDL, which dramatically increases the risk of the cardiovascular disease *atherosclerosis*. The resulting disorder is called *familial hypercholesterolemia* and is hereditary, as it propagates with the mutated gene.

2.3 Concluding Remarks

The dogma, if you will, of Systems Biology is that we must understand both the connectivity in systems and their self-organisation in order to understand biology [vMJ$^+$07]. Therefore this chapter has presented the main features of cellular information processing and cellular transport. In the course of the presentation we have identified two aspects, i.e., genetic transcription and receptor mediated endocytosis of Low Density Lipo-protein, that we shall abstractly model and subject to static analysis in later chapters.

Chapter 3

Modelling in Process Calculus

"By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race."
— Alfred North Whitehead

This chapter presents a variant of the BioAmbients calculus of Regev et al. [RPS$^+$04, Reg03, Car04], a sibling of Mobile Ambients (Cardelli and Gordon [CG00]) designed to model biological systems. The material is intended to serve multiple purposes. First of all it introduces the modelling formalism addressed by the technical developments of later chapters. It also introduces the basic concepts of process calculi, in particular a large family of languages that descend from the Calculus of Communicating Systems [Mil89, Mil99, CG00]. And finally it serves as an introduction to the 'cells-as-computation' abstraction that was pioneered by Regev et al [RPS$^+$04].

The BioAmbients calculus is very expressive and includes modelling primitives for many essential 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 the chemically active membrane-bound compartments that are ubiquitous in eukaryotic cells. In order to make this spatial abstraction operational, the calculus incorporates a set of (co-)capabilities that allow processes to alter the local nesting hierarchy.

Secondly, the calculus preserves 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. In order to make this part of the modelling formalism operational in the context of the ambient-as-compartment abstraction the corresponding set of (co-)actions allows communication across ambient boundaries as well as locally.

The resulting calculus is quite extensive in terms of modelling primitives. Also the set of control structures for processes is slightly larger than what is traditionally studied for Mobile Ambients, including non-deterministic (external) choice, as well as a general recursion construct, in the manner of CCS [Mil89]. This is needed in order to allow a precise modelling of biological phenomena.

What brings all of these elements together and completes the biological abstraction is a reaction semantics in the style of the Chemical Abstract Machine [BB90]. The interpretation provided by this semantics 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 *molecular complementarity*, i.e., expose *reactive domains* (modelled by prefixes) that are complementary in terms of *shape* (channel name) and *purpose* (matching capability/co-capability).

The chapter is in five sections. We start in Section 3.1, by giving an informal step-by-step introduction to a family of process calculi commonly used for the modelling of biological systems. Here we shall consider a notion of simple reactive processes in the style of CCS [Mil80], a notion of complex forming processes in the style of π-calculus [Mil99], and, finally, the class of compartment forming processes modelled by the BioAmbients calculus [RPS^{+}04]. Then, in Section 3.2, we formalise the reaction semantics of the BioAmbients calculus. After this we present a BioAmbients model of the LDL degradation pathway in Section 3.3 and a BioAmbients model of genetic transcription in Section 3.4. Finally, in Section 3.5, we summarise and relate to other approaches.

3.1 The BioAmbients Modelling Language

3.1.1 Simple Reactive Processes

In order to gently introduce the analogy between chemical reagents and computational reactive processes we start from the basics and consider a simple language of concurrent reactive process expressions. Besides the omission of internal actions this language is identical to Milner's Calculus of Communicating Systems (CCS) [Mil80], and forms the core of the BioAmbients modelling language.

As usual for process calculi the language incorporates two design elements:

Firstly, there is a set of *primitives*, each of which denotes the capability of performing some basic biological activity, which we consider as *atomic*. In the case of simple reactive processes the fundamental activity is to engage in reaction. As described in the previous chapter, reaction in the bio-molecular domain is primarily a matter of complementary physical shapes, the *binding sites* exposed by molecules, coming sufficiently close to each other as shown in Fig. 2.1. In the calculus we capture this by *constants*, n, m, that we pick from a denumerable set, $\mathcal{C}onst$, and use as abstract representatives of the interactions such facilitated. The sites that participate in reactions are physical complements of one another, i.e. if the one is convex then the other is concave. This leads to a notion of *simple capabilities* for reaction, where $n!$ and $n?$ denote the complementary binding sites that facilitate the interaction identified by n. Technically, we shall use standard terminology and call n a *channel* to point out that n denotes an (invisible) bond in the interaction topology of bio-chemical reactions.

Definition 3.1 (Simple Capabilities) A *simple capability* is an element of the set defined by the following syntax:

$$\begin{array}{rcrl} M \in \mathbf{Cap} & ::= & n! & \text{Reaction capability} \\ & | & n? & \text{Reaction co-capability} \end{array}$$

where $n \in \mathcal{C}onst$. ■

Secondly, there is a set of *control structures* that allow *process expressions* to be composed from the primitives in various ways. *Sequential process expressions* are the most basic of such compositions, and they are suitable for abstract descriptions of isolated reactive entities such as, e.g., proteins.

Definition 3.2 (Sequential processes expressions) A *sequential process* is an element of the set defined by the following syntax:

$$\begin{array}{rcrl} P \in \mathbf{Proc} & ::= & \sum_{i\in I} M_i^{\ell_i} \,.\, P_i & \text{Summation (guarded choice)} \\ & | & \mathsf{rec}\,X.\,P & \text{Recursive process (defining } X \triangleq \mathsf{rec}\,X.\,P\text{)} \\ & | & X & \text{Process identifier} \end{array}$$

where I is any finite indexing set. We shall use P, Q, R and their primed or sub-scripted variants (e.g., P') to denote process expressions.

Remark 3.3 (Labels) In preparation of the static analyses of later chapters we attach a label, $\ell_i \in \mathbf{Lab}$, to each capability prefix, M_i. It shall merely serve as a useful pointer into the process, and has no semantic significance. For any concrete process, P, we shall assume that **Lab** is a finite set.

Notation 3.4 It is customary to write concrete k-way instances of summation, i.e., $|I| = k$, as $M_1 \,.\, P_1 + \cdots + M_k \,.\, P_k$. Two special cases emerge when $k = 0$ or $k = 1$. They implicitly give rise to the following additional constructs:

$$\begin{array}{rclll} P \in \textbf{Proc} & ::= & & \ldots & \\ & | & & \mathbf{0} & \text{Inaction} \\ & | & & M^{\ell} \,.\, P & \text{Prefix} \end{array}$$

We shall often omit trailing occurrences of $\mathbf{0}$, i.e., simply write M rather than $M \,.\, \mathbf{0}$, when writing process expressions.

Definition 3.5 (Identifier binding) In $\mathsf{rec}\,\mathrm{X}.\,P$ the displayed occurrence of X is *identifier binding*, asserting that $X \triangleq P$, with *scope* P. An occurrence of a process identifier Y in a process is *bound* if it is, or lies within the scope of, a binding occurrence of Y. If not bound, Y is *free*. A process with no free identifiers is *identifier closed*.

Notation 3.6 (Identifier substitution) We shall write $P[{}^{Q}/_{X}]$ for the substitution of Q for X in P, i.e., for the process that is as P except that every free occurrence of the process identifier X is replaced by the process expression Q.

Remark 3.7 In the presence of recursive processes the operational definitions of *bound* and *free process identifiers* as well as *application of substitution* are not straightforward and we postpone the formal treatment until Chapter 5. ■

Informally, the expressions carry the following meaning:

Inaction, $\mathbf{0}$, 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*, $M^{\ell} \,.\, P$, denotes a process that is capable of participating in the reaction identified by the capability 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.

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 capabilities M_i. Again, exercising some M_j turns the process into the corresponding continuation P_j; the remaining capabilities 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.

The *recursive process*, $\mathsf{rec}\,\mathrm{X}.\,P$, models recurrent behaviour. Technically, the construct denotes a process that behaves as $P[^{\mathsf{rec}\,\mathrm{X}.\,P}/_X]$. Biologically, this corresponds to entities that have cyclic behaviour. One example is an entity $\mathsf{rec}\,\mathrm{X}.\,M^{\ell}\,.\,X$, such as an enzyme, that may participate in the same reaction – identified by M – over and over. Another example, that of a stateful entity $\mathsf{rec}\,\mathrm{X}.\,(\mathtt{off}?\,.\,\mathtt{on}?\,.\,X + M^{\ell}\,.\,X)$, 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.

Concurrent process expressions emerge when sequential (or concurrent) process expressions are combined by parallel composition; they are suitable for abstract descriptions of *reactive systems* such as, e.g., bio-chemical solutions.

Definition 3.8 (Concurrent processes) A *concurrent process* is an element of the set defined by the following syntax:

$P \in \mathbf{Proc}$	$::=$	$(n)\,P$	Name restriction
	$\mid$	$P \mid P$	Parallel composition
	$\mid$	$\sum_{i\in I} M_i^{\ell_i}\,.\,P_i$	Summation (guarded choice)
	$\mid$	$\mathsf{rec}\,\mathrm{X}.\,P$	Recursive process (defining $X \triangleq \mathsf{rec}\,\mathrm{X}.\,P$)
	$\mid$	X	Process identifier

where I is any finite indexing set.

Definition 3.9 (Constant binding) In $(n)\,P$ the displayed occurrence of the constant n is *binding* with *scope* P. ■

The language extends that of sequential process expressions. Informally the new constructs carry the following meaning:

The *parallel composition*, $P \mid Q$, denotes the concurrent composition of processes P and Q. In the resulting *reactice system* reactions may occur between P and Q if the one exposes a capability $n!$ and the other the corresponding co-capability $n?$. For example, the system $n!\,.\,P' \mid n?\,.\,Q'$ may react and become $P' \mid Q'$. 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.

Finally, in the *name restriction*, $(n)\,P$, the scope of the name n is restricted 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 notion of confinement; in the context of concurrent process expressions the correspondence is rather weak because it is not straightforward how to model the dynamics of either physical confinements, such as compartments, or virtual ones, such as complexes. In the former case n would simple correspond to a type of reaction only taking place in a particular compartment, in the latter to a reaction facilitated by binding sites that are exposed toward the interior of a particular coordination compound only.

Remark 3.10 In the context of parallel composition the notion of *replication* arises as a special case of the recursive process. Intuitively. $\mathsf{rec}\, \mathrm{X}.\, P \,\big|\, X$ denotes a potentially unbounded supply of instances of P. This is usually written $!P$. ■

3.1.2 Complex Forming Processes

In order to strengthen the analogy between chemical reagents and computational processes we now extend the language into one of *complex forming process expressions*. Save the omission of internal actions and conditionals, this language is identical to the π-calculus of Milner, Parrow, and Walker [MPW92, Mil99, SW01, Par01]. Suitable for the modelling of biochemical entities and complexes it forms a powerful subset of the BioAmbients calculus.

Complex forming processes emerge when we allow names to be communicated when reactions occur. This is embodied in the *local communication capabilities*, a new set of primitives that subsumes the simple capabilities as a special case. The definition relies on a notion of *variables*, p, q, that we pick from a denumerable set, $\mathcal{V}ar$, and use as placeholders for the constants received during communication. As we shall see, the syntax admits both constants and variables in many positions; hence it is common to refer to them simply as *names*, x, y, and pick them from the denumerable set, $\mathcal{N}ame$, that is composed as the disjoint union of constants and variables, i.e., $\mathcal{N}ame = \mathcal{C}onst \uplus \mathcal{V}ar$.

Definition 3.11 (Local communication capabilities) By a *local communication capability* we shall understand an element of the set defined by the following syntax:

$$\begin{array}{rcll} M \in \mathbf{Cap} & ::= & x!\{y\} & \text{Local output capability} \\ & | & x?\{p\} & \text{Local input (co-)capability} \end{array}$$

Definition 3.12 (Variable binding) In $x?\{p\}\,.\,P$ the displayed occurrence of the variable p is *binding* with *scope* P.

Definition 3.13 (Free and bound names) An occurrence of a name x in a process is *bound* if it is, or lies within the scope of, a binding occurrence of x. If not bound, x is *free.* A process with no free name is *name closed.*

Notation 3.14 (Name substitution) We shall write $P[{}^{m}/_{x}]$ for the substitution of the constant m for the name x in the process expression P, i.e., the process that is as P except that every free occurrence of the name x is replaced by the constant m.

Remark 3.15 Again, we postpone the operational definitions of *bound* and *free names* until Chapter 5. ■

As was the case for the simple capabilities, the local communication capabilities only become operational when used for prefixing. Because of the involved name passing, however, reactions are no longer symmetric activities between processes; rather they require one of the involved processes to act as sender and the other to act as recipient: The prefix $n!\{m\} . P$ is a sender; it denotes a process that is capable of participating in the reaction identified by n, and if this capability is exercised the process shares its knowledge about the binding of m with the reaction partner and continues as P. In a complementary fashion the prefix $n?\{p\} . Q$ is a recipient; it denotes a process that is capable of participating in the reaction identified by n, and if this capability is exercised the process learns about some name binding, e.g., m, that is then substituted for p throughout Q, such that the process continues as $Q[{}^{m}/_{p}]$ rather than Q.

Thus, the extension enables the passing of values from one entity to the other. However, the main strength lies in the notion of *scope extrusion* that accompanies this ability. For example, in a system $(m)\, n!\{m\} . P \mid n?\{p\} . Q \mid P'$ a local communication reaction might occur such that in the resulting system $(m)\,(P \mid Q[{}^{m}/_{p}]) \mid P'$ the scope of m has been *extruded* to include both P and $Q[{}^{m}/_{p}]$, i.e., P and $Q[{}^{m}/_{p}]$ now share the bond denoted by m, but it is not shared by anyone else.

This can be used to model the biological notion of complexes — especially the, often temporary, coordination compounds of molecular machineries that occur in conjunction with, e.g., the transcription of genes or the translation of RNA. These compounds often form in an ad-hoc manner in order to perform some non-atomic piece of work, after which they disassemble again. The ad-hoc nature of such a compound calls for individual modelling of their constituting components. The fixed cooperation structure of the compound, however, requires the model to enforce a temporary 'lock' on the set of constituents throughout the lifetime of a complex. This is exactly what scope extrusion and recursion provides.

Now, when a communication reaction takes place, e.g., in $n!\{m\}\,.\,P \,\big|\, n?\{p\}\,.\,Q$, then the result is defined in terms of a substitution $P \,\big|\, Q[{}^{m}/_{p}]$ of a constant for a variable. If m is bound in Q, i.e., Q contains a sub-expression $(m)\,Q'$, then m becomes spuriously bound by this binder if Q' is of the form $\cdots p \cdots$.

This is called *constant capture* and, in order to avoid it, it is usual to demand *static scope* and define a notion of *α-equivalence* that allows bound constants to be freely changed.

Definition 3.16 (α-equivalence) A *change of bound constants* or *α-renaming* within a process P is the replacement of a subterm $(n)\,Q$ of P by $(m)\,Q[{}^{m}/_{n}]$ where $m \notin \mathsf{fn}(Q)$. It is common to say that P and Q are *α-equivalent*, $P \equiv_\alpha Q$, if Q can be obtained from P by a finite number of changes of bound constants.

Remark 3.17 Communication is defined in terms of a substitution, $P[{}^{n}/_{x}]$, of a constant for a name, and α-equivalence is defined in terms of a substitution, $P[{}^{m}/_{n}]$, of a constant for a constant. In contrast we shall ensure in Chapter 5 that the need of substituting a variable for a variable or a process identifier for a process identifier shall never arise; hence we define α-equivalence neither for variables nor process identifiers.

Remark 3.18 (Canonical names) Many of the technical developments of the following sections and chapters will rely on the ability to statically track the names that occur in the processes of interest. Due to α-equivalence, however, the direct syntactical representation of names is not stable under the semantic relations; hence ordinary names are not suitable for carrying static analysis information.

We solve this issue in the manner that is standard for Flow Logic based static analysis, e.g., [NNP04, HJNN99, BDNN01]. We associate each name x with a *canonical name* $\lfloor x \rfloor \in \mathbf{Name}$ and demand that α-renaming be *disciplined*, i.e., performed in such a way that canonical names are preserved, even when the syntactical representations change. When N is a set of names we shall write $\lfloor N \rfloor$ to denote the point-wise extension of $\lfloor x \rfloor$. Similarly we extend the operator to capabilities, i.e., $\lfloor M \rfloor$ is M where every name is canonicalised, and processes, i.e., $\lfloor P \rfloor$ is P where every name is canonicalised.

For any concrete process, P, we shall assume that $\mathbf{Name}$, in contrast to $\mathcal{N}ame$, is a finite set. And, like the ordinary names, this set is composed as the disjoint union of the canonical constants, $\lfloor n \rfloor \in \mathbf{C}$, and the canonical variables, $\lfloor p \rfloor \in \mathbf{V}$, i.e., $\mathbf{Name} = \mathbf{C} \uplus \mathbf{V}$.

In practice we are going to take $\mathbf{Name} \subset \mathcal{N}ame$. In the case of variables, which are not subject to α-conversion, it then suffices to choose $\lfloor p \rfloor = p$. In the case of

$P \in \mathbf{Proc}$	$::=$	$(n)\,P$	Name restriction
	$\mid$	$[\,P\,]^{\mu}$	Ambient boundary
	$\mid$	$P \mid P'$	Parallel composition
	$\mid$	$\sum_{i \in I} M_i^{\ell_i}\,.\,P_i$	Summation (guarded choice)
	$\mid$	$\mathsf{rec}\,X.\,P$	Recursive process (defining $X \triangleq \mathsf{rec}\,X.\,P$)
	$\mid$	X	Process identifier

Table 3.1: The syntax of BioAmbients processes, P.

constants, however, we shall assume that each distinct constant, n, occurring in the initial static program code gives rise to an equivalence class of constants represented by the corresponding canonical constant, $\lfloor n \rfloor$. Disciplined α-renaming then demands that the α-renaming process always picks the replacement name from the equivalence class of the replaced name and that the corresponding α-equivalence (Table 3.3) only holds when the bound names are indeed from the same class. ■

3.1.3 Compartment Forming Processes

As noted by Regev [Reg03] membrane bounded compartments may be modelled as complex forming processes, but the underlying analogy is quite weak. Thus she proposed a better alternative in the form of BioAmbients. In doing so she proposed that *compartment forming processes* emerge from complex forming processes, when you allow spatial boundaries to be explicitly modelled.

Definition 3.19 (BioAmbients processes) The set of *BioAmbients process expressions* is defined by the syntax of Table 3.1.

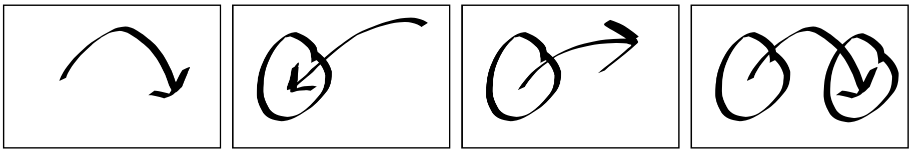
Notation 3.20 We use the heavy brackets, $[\!$and$\,]$, to represent ambient boundaries; the ordinary brackets, [and], are reserved for the notion of *substitution*.

Remark 3.21 (Roles) In preparation for the static analyses of later chapters we attach a role, $\mu \in$ **Role**, to each ambient boundary. It shall merely serve as a useful pointer into the process, and has no semantic significance. For any concrete process, P, we shall assume that **Role** is a finite set. ■

This extends the language of complex forming processes with the *ambient boundary* construct, $[\,P\,]^{\mu}$, which denotes a process P encapsulated by a spatial boundary. There is an obvious correspondence to the biological concept of a compartment, which is exactly a physical enclosure that creates a strong spatial separation between inside and outside. In some instances, however, it is also

$M \in$ **Cap**	$::=$					**Communication capabilities**
		$x!\{y\}$	$\mid$	$x?\{p\}$		local communication
	$\mid$	$x\_!\{y\}$	$\mid$	$x\text{\^{}}?\{p\}$		parent to child communication
	$\mid$	$x\text{\^{}}!\{y\}$	$\mid$	$x\_?\{p\}$		child to parent communication
	$\mid$	$x\#!\{y\}$	$\mid$	$x\#?\{p\}$		sibling to sibling communication
						Movement capabilities
	$\mid$	enter x	$\mid$	accept x		enter movement
	$\mid$	exit x	$\mid$	expel x		exit movement
	$\mid$	merge– x	$\mid$	merge+ x		merge movement

Table 3.2: Syntax of BioAmbients capabilities, M.



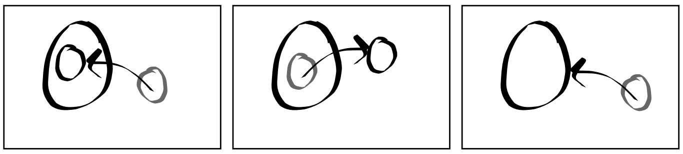
(a) local (b) parent to child (c) child to parent (d) sibling

Figure 3.1: Communication styles of the BioAmbients calculus.

useful for the modelling of 'hard' compounds, i.e., complexes that are not formed as temporary coordination compounds. Finally, it may model non-membranal, enclosure forming, biological entities, such as the clathrin coats that force the formation of internalisation vesicles by the plasma membrane (Section 2.2.3).

Definition 3.22 (BioAmbients capabilities) The set of *BioAmbients capabilities* is defined by the syntax of Table 3.2. ■

The pair $x\_!\{y\}/x\text{\^{}}?\{p\}$ causes *parent to child communication*, i.e., a process may communicate a message to a process encapsulated by a neighbouring ambient (Fig. 3.1(b)); this may correspond to a variety of receptor mediated interactions where the co-action is perceived as a surface receptor of a compartment.



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

Figure 3.2: Movement styles of the BioAmbients calculus.

In contrast, the pair $x\hat{}!\{y\}/x\_?\{p\}$ causes *child to parent communication*, i.e., a process may communicate a message to a process that is a neighbour of its immediately enclosing ambient (Fig. 3.1(c)); biologically this could e.g. correspond to interaction between a component of a complex and a molecule neighbouring the complex.

Finally, the pair $x\#!\{y\}/x\#?\{p\}$ allows *sibling to sibling communication*, a process within some ambient may communicate a message to a process located within an ambient neighbouring the first (Fig. 3.1(d)); biologically this corresponds, e.g., to receptor mediated interaction between a complex and a compartment, communication between components of different complexes, or inter-compartment (e.g., hormonal) signalling.

In the case of movement capabilities the effect of reaction is that the local hierarchy of nested ambients changes.

The effect of an interaction between enter x and accept x is that one ambient enters a neighbouring one (Fig. 3.2(a)); biologically this corresponds, e.g., to *endocytosis* where a compartment (selectively) subsumes another large entity.

The capabilities exit x and expel x cause one ambient to leave the immediately enclosing one (Fig. 3.2(b)); this corresponds, e.g., to *exocytosis* where a compartment (selectively) secretes some matter.

Finally, merge+ x and merge– x simply make two neighbouring ambients merge (Fig. 3.2(c)); this corresponds to fusion of biological compartments.

Example 3.23 Our running example shall be the following program, P_{eat}, which is inspired by the production of metabolites by catabolism of nutrients; as we shall explain later it models how 'food particles' (nutrients) may either be ignored, digested, or emitted as secretion by the cell:

$$
\begin{array}{l}
(rj)\,(ac)\,(rea)\,(RL) \\
[\ (\quad \mathsf{rec}\,\mathrm{Z}.\,\mathsf{expel}\ rj^{1}\,.\,\mathrm{Z} \\
\qquad \|[\ (\quad \mathsf{rec}\,\mathrm{Y}.\,(\ rea\hat{}?\{rl\}^{2}\,.\,\mathsf{expel}\ rl^{3}\,.\,\mathrm{Y} \\
\qquad\qquad\qquad\qquad +\mathsf{exit}\ rj^{4}\,.\,\mathrm{Y} \\
\qquad\qquad\qquad\qquad +\mathsf{enter}\ ac^{5}\,.\,\mathrm{Y}\ ) \\
\qquad\qquad \|[\ \mathsf{exit}\ RL^{6}\,.\,\mathbf{0}\ \ ]^{nutrient}\ \ )\,]^{food} \\
\qquad \|[\quad \mathsf{rec}\,\mathrm{S}.\,(\ rea\_!\{RL\}^{7}\,.\,\mathrm{S} \\
\qquad\qquad\qquad +\mathsf{expel}\ rj^{8}\,.\,\mathrm{S} \\
\qquad\qquad\qquad +\mathsf{accept}\ ac^{9}\,.\,\mathrm{S}\ )\,]^{cell}\ \ )\,]^{system}
\end{array}
$$

$\square$

3.2 Semantics of BioAmbients

The notion of Reaction Semantics is inspired by Berry and Boudol's Chemical Abstract Machine [BB90], which pioneered the view that concurrent process expressions are 'really' chemical solutions of reactive entities. These solutions are heated and stirred by an invisible device, and entities may react if they come close and exhibit molecular complementarity. Being extremely appealing, this abstraction is now widely used in the context of process calculi. In particular, it is the traditional choice for ambient calculi. Thus, when conceiving of BioAmbients, Regev [RPS$^+$04, Reg03] gave the language a reaction semantics, which 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, $\xrightarrow{\tilde{\ell}}$.

Definition 3.24 (Heating relation) The *heating relation*, $\Rrightarrow$, is the least binary relation on **Proc** that is inductively defined by the axioms and rules of Table 3.3. 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, $\Rrightarrow$).

Remark 3.25 (Structural congruence) The ordinary structural congruence relation, $\equiv$, of BioAmbients [Reg03] is nearly fully embedded in the heating relation, where we write $\equiv$ for the conjunction of $\Rrightarrow$ and $\Lleftarrow$. 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.

Remark 3.26 (Unfolding of recursive processes) We use the heating relation to unfold recursive processes. Another choice is usually associated with Structural Operational Semantics of process calculi [Mil99], where it is custom to have the following inference rule:

$$\frac{P[{}^{\mathsf{rec}\,\mathrm{X}.\,P}/_{X}] \xrightarrow{\alpha} Q}{\mathsf{rec}\,\mathrm{X}.\,P \xrightarrow{\alpha} Q}$$

Scope of restrictions:

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

H-SAMB $(n)\,([\,P\,]^{\mu}) \equiv [\,(n)\,P\,]^{\mu}$

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

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 $(P + Q) + R \equiv P + (Q + R)$

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 $(n)\,\mathbf{0} \Rrightarrow \mathbf{0}$

Unfolding of recursive processes:

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

Reflexivity and transitivity:

H-REFL $P \Rrightarrow P$

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

Congruence:

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

H-CAMB $$\frac{P \Rrightarrow Q}{[\,P\,]^{\mu} \Rrightarrow [\,Q\,]^{\mu}}$$

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

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

Table 3.3: The heating relation $P \Rrightarrow Q$ on processes. We write $P \equiv Q$ when both $P \Rrightarrow Q$ and $P \Lleftarrow Q$.

This works fine in the presence of a reaction rule

$$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

that allows two recursive processes to be simultaneously unfolded. In the presence of a reaction semantics, however, this does not work, and we delegate unfolding to the heating relation in order to allow simultaneous unfolding. ■

The heating relation serves both as a stirrer, cleaner, and catalyst, much like real heat does to a real chemical solution. The rules and axioms have the following meaning:

The rules for *scope of restrictions* ensure that a name restriction is (almost) completely free to migrate in and out of ambient boundaries, parallel compositions, other name restrictions. This guarantees that a name restriction cannot spuriously block reactions, because sooner or later it will migrate *out* of the way. Note, that together with α-conversion (application of H-ALPH) this axiom implicitly captures the aforementioned notion of *scope extrusion*.

The rules for *reordering of parallel processes* make systems behave like multisets of process expression, i.e. solutions of reactive entities. Similarly, the *reordering rules for summations* make summations behave as (subordinate) multisets, i.e., dynamic entities that spin and thereby make their binding sites accessible from all directions.

The notion of *α-equivalence* ensures that solutions are indistinguishable up to disciplined changes of bound names. Thus the concrete representation of a particular channel is immaterial, as long as the relevant reactive entities agree on it.

The H-NEVA rule serves to eliminate inactive processes. This corresponds to Berry and Boudol's notion of *inaction cleanup* and biologically we shall think of this as *evaporation*.

The H-RDIF rule serves to eliminate useless restrictions. This corresponds to Berry and Boudol's notion of restriction cleanup and biologically we shall think of this as *diffusion* or secretion of garbage.

The rule for *unfolding of recursive processes* unfolds a recursive definition once, thereby releasing a single instance of the body into the system. This corresponds to Berry and Boudol's fixpoint rule and biologically we shall think of this as a process that is catalysed by heating.

The rules for *reflexivity* and *transitivity* ensure that the heating of a system may cause any number, that is zero or more, concrete stirring, cleaning or unfolding actions. Note that symmetry of $\equiv$ follows implicitly.

Finally, the rules for *congruence* ensure that the equivalence extends to all

Movement:

R-ENT $$[\,(\mathsf{enter}\ n^{\ell_1}\,.\,P+P')\mid P''\,]^{\mu_1}\mid[\,(\mathsf{accept}\ n^{\ell_2}\,.\,Q+Q')\mid Q''\,]^{\mu_2}\xrightarrow{(\ell_1,\ell_2)}[\,[\,P\mid P''\,]^{\mu_1}\mid Q\mid Q''\,]^{\mu_2}$$

R-EXT $$[\,[\,(\mathsf{exit}\ n^{\ell_1}\,.\,P+P')\mid P''\,]^{\mu_1}\mid(\mathsf{expel}\ n^{\ell_2}\,.\,Q+Q')\mid Q''\,]^{\mu_2}\xrightarrow{(\ell_1,\ell_2)}[\,P\mid P''\,]^{\mu_1}\mid[\,Q\mid Q''\,]^{\mu_2}$$

R-MRG $$[\,(\mathsf{merge}{-}\ n^{\ell_1}\,.\,P+P')\mid P''\,]^{\mu_1}\mid[\,(\mathsf{merge}{+}\ n^{\ell_2}\,.\,Q+Q')\mid Q''\,]^{\mu_2}\xrightarrow{(\ell_1,\ell_2)}[\,P\mid P''\mid Q\mid Q''\,]^{\mu_2}$$

Communication:

R-L2L $$(n!\{m\}^{\ell_1}\,.\,P+P')\mid(n?\{p\}^{\ell_2}\,.\,Q+Q')\xrightarrow{(\ell_1,\ell_2)}P\mid Q[m/p]$$

R-P2C $$(n\_!\{m\}^{\ell_1}\,.\,P+P')\mid[\,(n\hat{}?\{p\}^{\ell_2}\,.\,Q+Q')\mid Q''\,]^{\mu}\xrightarrow{(\ell_1,\ell_2)}P\mid[\,Q[m/p]\mid Q''\,]^{\mu}$$

R-C2P $$[\,(n\hat{}!\{m\}^{\ell_1}\,.\,P+P')\mid P''\,]^{\mu}\mid(n\_?\{p\}^{\ell_2}\,.\,Q+Q')\xrightarrow{(\ell_1,\ell_2)}[\,P\mid P''\,]^{\mu}\mid Q[m/p]$$

R-S2S $$[\,(n\#!\{m\}^{\ell_1}\,.\,P+P')\mid P''\,]^{\mu_1}\mid[\,(n\#?\{p\}^{\ell_2}\,.\,Q+Q')\mid Q''\,]^{\mu_2}\xrightarrow{(\ell_1,\ell_2)}[\,P\mid P''\,]^{\mu_1}\mid[\,Q[m/p]\mid Q''\,]^{\mu_2}$$

Execution in context:

R-RES $$\frac{P\xrightarrow{\tilde{\ell}}Q}{(n)\,P\xrightarrow{\tilde{\ell}}(n)\,Q}$$ R-AMB $$\frac{P\xrightarrow{\tilde{\ell}}Q}{[\,P\,]^{\mu}\xrightarrow{\tilde{\ell}}[\,Q\,]^{\mu}}$$ R-PAR $$\frac{P\xrightarrow{\tilde{\ell}}Q}{P\mid R\xrightarrow{\tilde{\ell}}Q\mid R}$$

Stirring, cleaning, and catalysis:

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

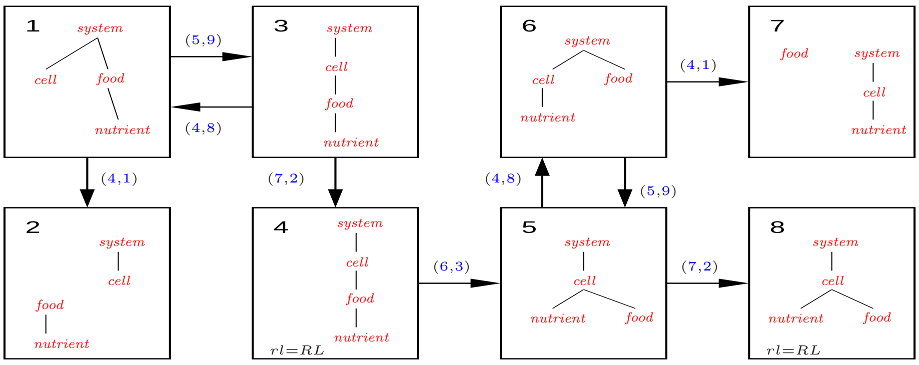
Table 3.4: The reaction relation, $P\xrightarrow{\tilde{\ell}}Q$, on processes.

relevant sub-expressions, i.e., two process expressions denote the same solution if all sub-expressions denote the same sub-ordinate solutions.

Definition 3.27 (Reaction relation) The *reaction relation* is the least binary relation on **Proc** defined inductively by the axioms and rules of Table 3.4. When it holds between two processes P and Q, written $P\xrightarrow{\tilde{\ell}}Q$ where

$\tilde{\ell} = (\ell_1, \ell_2)$ is a pair of labels, 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.

Example 3.28 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. $\square$

Remark 3.29 (Semantic labels) The labels, ℓ, that we attach to the semantic arrow, $\xrightarrow{\tilde{\ell}}$, have no semantic significance. They simply constitute an *instrumentation* that helps us define a *collecting semantics*, $\xrightarrow{\tilde{L}}{}^{\star}$, i.e., the reflexive transitive closure of $\xrightarrow{\tilde{\ell}}$, defined as follows:

$$P_\star \xrightarrow{\varepsilon}{}^{\star} P_\star \qquad\qquad \frac{P_\star \xrightarrow{\tilde{L}}{}^{\star} P \quad P \xrightarrow{\tilde{\ell}} Q}{P_\star \xrightarrow{\tilde{L}\tilde{\ell}}{}^{\star} Q}$$

Thus, for a given derivation, $P \xrightarrow{\tilde{L}}{}^{\star} Q$, the string $\tilde{L}$ constitutes a record, in the form of a *trace*, of the transitions that have come to pass. We shall use this in the sequel, when formulating our correctness results, and quite often 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 reaction relation constitutes our definition of the reactive behaviour of processes. Every axiom has two ingredients. In the case of movement reactions the local ambient hierarchy is changed and in the case of communication reactions 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 summands are discarded.

The movement reactions are those of *enter movement*, *exit movement*, and *merge movement*, defined by axioms R-ENT, R-EXT, and R-MRG, respectively, and they enable the movement behaviours described in Section 3.1.3.

The communication reactions are those of *local communication*, *parent to child communication*, *child to parent communication*, and *sibling to sibling communication*, defined by axioms R-L2L, R-P2C, R-C2P, and R-S2S, respectively, and they enable the communication behaviours described in Section 3.1.3.

The rules for *execution in context* ensures that reactivity extends to all relevant parts of the system. This asserts that sub-expressions of restrictions, ambients, and parallel compositions are indeed reactive and, thus, denote subordinate solutions.

Finally, the rule for *cleaning, stirring, and catalysis* allows systems to be stirred and heated before and after reactions. This enables the view that reaction is a relation on warm and well-stirred solutions rather than fixed algebraic expressions.

Remark 3.30 (Unrestricted sums vs. guarded choice) Note that we omit the unrestricted choice, $P + Q$, of the original language [Reg03] in favour of the more conventional guarded sum, $\sum_{i \in I} M_i \,.\, P_i$. We do this because unrestricted choice a) is difficult to handle in a reaction semantics in the presence of general recursion [PNN06a], and b) seems to provide no *essential* extension of expressive power. The point a) is of relevance to the developments in Chapter 8, where a well-defined kill component is extremely difficult to specify in the presence of unrestricted choice. ■

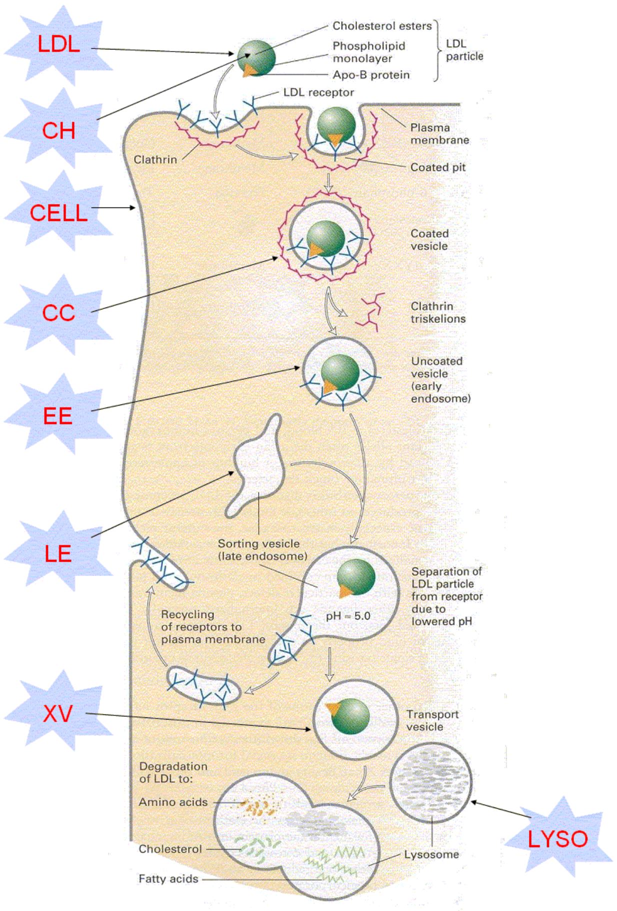


Figure 3.3: Annotated LDL degradation pathway [LBZ+99].

3.3 CASE: Modelling the LDL Degradation Pathway

We now turn to the modelling of the LDL degradation pathway (Section 2.2.3), which is the first of our two case studies. Receptor mediated endocytosis is primarily a transport scenario. It involves both transmembranal coordination and compartment based transport; this is modelled by ambients that communicate and move.

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. Thus, we have the following correspondences:

- The LDL role models LDL particles.
- The EE role models early endosomes (true compartments).
- The LE role models late endosomes (true compartments).
- The CC role models clathrin coats (coating the EE in the coated vesicle).
- The $LYSO$ role models lysosomes (true compartments).
- The $CELL$ role models cells (true compartments).
- The XV role models transfer vesicles (true compartments).
- The CH role models cholesterol.

When we can do so without ambiguity, we will use the abbreviated 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 LDL binding but ignores the details of receptor recycling. Nonetheless this allows the analysis to highlight certain medical issues.

In Nature each compartment and reaction would be present in the thousands. For the purposes of this dissertation, however, we are merely interested in qualitative information and, hence, it suffices for us to model a single representative for each biological entity.

In Table 3.3 the LDL (described in LipoProtein) is initially located outside of the $CELL$ (described in Cell). 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 $CELL$.

At this stage the early endosome has not been formed yet. We model, however, the membrane patch and the transmembranal LDL receptors, which are later going to fold into the early endosome, as a process capable of evolving into the EE ambient. As explained in Section 2.2.3, the clathrin coated early endosome may be formed with or without bound LDL particles. We model this by a summation that demands one of the following three binding scenarios to occur before the EE ambient is released:

$$
\begin{array}{l}
\mathsf{LipoProtein} = \\
\quad [\ LDLrcpt\#!\{ApoB\}^{1} \,.\, \mathsf{enter}\ ApoB^{2} \,.\, \mathsf{enter}\ ee^{3} \,.\, \mathsf{enter}\ xv^{4} \,.\, proc\hat{}?\{Hydr\}^{5} \,. \\
\qquad\quad (\ \mathsf{expel}\ Hydr^{6} \,.\, \mathbf{0} \\
\qquad\qquad \mid [\ \mathsf{exit}\ Hydr^{7} \,.\, \mathbf{0}\ ]^{CH}\ )\ ]^{LDL} \\
\mathsf{Endo} = \\
\quad \mathsf{enter}\ AP2^{23,16,9} \,.\, \mathsf{exit}\ AP2^{24,17,10} \,.\, \mathsf{merge-}\ Le^{25,18,11} \,.\, \mathbf{0} \\
\mathsf{EarlyEndo} = \\
\quad [\ \mathsf{accept}\ ee^{22,15} \,.\, \mathsf{Endo}\ ]^{EE} \\
\mathsf{ClathrinCoat} = \\
\quad [\ EErcpt\hat{}?\{ap2\}^{26} \,.\, \mathsf{accept}\ ap2^{27} \,.\, \mathsf{expel}\ ap2^{28} \,.\, \mathbf{0}\ ]^{CC} \\
\mathsf{XferVesicle} = \\
\quad [\ (\ \mathsf{accept}\ xv^{31} \,.\, \mathbf{0} \\
\qquad\quad \mid \mathsf{exit}\ Le^{32} \,.\, \mathsf{merge-}\ lyso^{33} \,.\, \mathbf{0}\ \ )\ ]^{XV} \\
\mathsf{LateEndo} = \\
\quad [\ (\ \mathsf{merge+}\ Le^{29} \,.\, \mathsf{expel}\ Le^{30} \,.\, \mathbf{0} \\
\qquad\quad \mid \mathsf{XferVesicle}\ )\ ]^{LE} \\
\mathsf{Lysosome} = \\
\quad [\ \mathsf{merge+}\ lyso^{34} \,.\, proc\_!\{hydr\}^{35} \,.\, \mathbf{0}\ ]^{LYSO} \\
\mathsf{Cell} = \\
\quad [\ (\ (\ \ EErcpt\_!\{AP2\}^{8} \,.\, [\ \mathsf{Endo}\ ]^{EE} \\
\qquad\qquad +EErcpt\_!\{AP2\}^{12} \,.\, LDLrcpt\#?\{apob\}^{13} \,.\, \mathsf{accept}\ apob^{14} \,.\, \mathsf{EarlyEndo} \\
\qquad\qquad +LDLrcpt\#?\{apob\}^{19} \,.\, \mathsf{accept}\ apob^{20} \,.\, EErcpt\_!\{AP2\}^{21} \,.\, \mathsf{EarlyEndo}\ ) \\
\qquad\quad \mid \mathsf{ClathrinCoat} \\
\qquad\quad \mid \mathsf{LateEndo} \\
\qquad\quad \mid \mathsf{Lysosome}\ )\ ]^{CELL} \\
\\
(LDLrcpt)\,(EErcpt)\,(ApoB)\,(AP2)\,(ee)\,(cc)\,(lyso)\,(xv) \\
\qquad (Le)\,(proc)\,(hydr)\,(ap2)\,(ap) \\
(\ \ \mathsf{LipoProtein} \mid \mathsf{Cell}\ )
\end{array}
$$

Table 3.5: Abstract model of the LDL degradation pathway.

1. The extra-cellular part $LDLrcpt$ of the LDL receptor binds the $ApoB$ signal of the LDL thus forcing LDL to enter the $CELL$. Subsequently the intra-cellular part $EErcpt$ of the receptor is bound by the $AP2$ domain exposed by the CC bound adaptin molecules.

2. The intra-cellular part $EErcpt$ of the receptor is bound by the $AP2$ domain exposed by the CC bound adaptin molecules. Subsequently the extra-cellular part $LDLrcpt$ of the LDL receptor binds the $ApoB$ signal of the LDL thus forcing LDL to enter the $CELL$.

3. The intra-cellular part $EErcpt$ of the receptor is bound by the $AP2$ domain of the CC and the extra-cellular part $LDLrcpt$ is never bound.

If the LDL is in place inside the $CELL$ after the binding scenario it may enter the EE otherwise not. Either way, the internalisation of the clathrin coated pit may be completed by the EE entering the CC.

In Nature this internalisation process is atomic since the

$$[[[[[\ ]^{CH}]^{LDL}]^{EE}]^{CC}]^{CELL}$$

(or $[[[\ ]^{EE}]^{CC}]^{CELL}$) configuration arises instantaneously when the coated vesicle is completed and internalised. By modelling this as a sequence of events we are introducing *modelling artifacts* as is indeed a common phenomenon when using process algebras for describing biological systems. Most importantly, for the LDL to enter the EE we have to allow it into the $CELL$, which is biologically unsound. Thus, we must keep in mind, when interpreting the analysis results, that the EE must both enter and leave, the latter corresponding to the internalised early endosome shredding its clathrin coat, the CC in order to enter the $CELL$.

When released in the $CELL$ the EE is able to merge with the LE. This releases the LDL into the LE from where it may enter the XV. After the merge the XV is free to leave the LE with, or without, the LDL cargo.

Finally, the XV may merge with the $LYSO$, thus releasing the LDL cargo into its final destination where it may be hydrolysed into CH.

3.4 CASE: Modelling Genetic Transcription

In a similar manner we shall also fashion an abstract model of genetic transcription (Section 2.1.2), which is the second of our case studies. Genetic transcription is primarily a coordination scenario. It involves both the formation of coordination compounds and recurrent behaviour; this is modelled by recursive processes that coordinate within extruded scopes.

The resulting model has three components:

The genome is a collection of genes. Each gene repeatedly allows polymerase to attach at the promoter and then transcribe the gene by reading the nucleotides of the sequence one at a time. Hence, in the model we portray a gene as a recurrent process gene_i where each recurrence corresponds to a complete transcription and has three phases:

Initiation The gene supports the formation of coordination compounds by

```
gene1 =
    rec $G1 . (g1)\,(b1)\,(d1)$
    $tr!\{g1\}^{1} . g1!\{b1\}^{2} . g1!\{a\}^{3} .$
        $( \; g1!\{c\}^{4} . ( \; g1!\{a\}^{5} . ( \; g1!\{c\}^{6} . b1!\{d1\}^{7} . G1$
                                        $+ b1?\{d11\}^{8} . G1 \; )$
                            $+ b1?\{d11\}^{9} . G1 \; )$
          $+ b1?\{d11\}^{10} . G1 \; )$
genome =
  gene1
ATP =
    rec $A . ( \; a!\{a\}^{11} . \mathbf{0} \mid A \; )$
CTP =
    rec $C . ( \; c!\{c\}^{12} . \mathbf{0} \mid C \; )$
GTP =
    rec $G . ( \; g!\{g\}^{13} . \mathbf{0} \mid G \; )$
UTP =
    rec $U . ( \; t!\{t\}^{14} . \mathbf{0} \mid U \; )$
nTP =
  ( ATP | CTP | GTP | UTP )
polymerase =
    rec $P . tr?\{gene\}^{15} . gene?\{b\}^{16} .$  rec $X . ( \; gene?\{p\}^{17} . ( \; b!\{d\}^{18} . P$
                                                                   $+ p?\{dp\}^{19} . X \; )$
                                                     $+ b?\{dp\}^{20} . ( \; [\,\mathbf{0}\,]^{MRNA} \mid P \; ) \; )$

$(a)\,(c)\,(g)\,(t)\,(tr)\,(d)$
( genome | nTP | polymerase )
```

Table 3.6: Abstract model of genetic transcription.

offering scope extrusion of the private channels g_i and b_i on the public channel tr.

Elongation The gene supports a sequential 'reading' of its nucleotide sequence by offering a sequence of corresponding communications on the channel g_i. At each step the gene is prepared for the coordination compound to be broken up via a message on the private channel b_i.

Termination If all nucleotides are 'read' the gene breaks up the coordination compound by sending a message on the private channel b_1.

Note that, in the concrete example, the genome is a collection of only a single gene. As the BioAmbients syntax does not allow empty synchronising communications we use the d_i as dummy messages.

The cell provides a potentially unbounded supply of nucleoside tri-phosphates that provides the nucleotides for the growing messenger RNA. In the model we capture this by replicating processes corresponding to each of the NTPs in question. Each replica offers just a single interaction on a public channel corresponding to the specific nucleotide. In the concrete model the nTP is simply an unbounded collection of ATP, GTP, CTP, and UTP.

Polymerase is an enzyme that makes RNA copies of DNA templates. It repeatedly attaches to the promoter region of a (not specific) gene and then synthesises a corresponding RNA replica by processing one nucleotide at the time. Here we model the enzyme as a recurrent process, $\mathsf{polymerase}$, where each major recurrence, $\mathsf{rec}\,\mathrm{P}.\,\cdots$, corresponds to a complete transcription and each minor recurrence, $\mathsf{rec}\,\mathrm{X}.\,\cdots$, corresponds to a single nucleotide. Thereby we achieve a generic polymerase that will transcribe a gene of any length in three phases:

Initiation The polymerase forms a coordination compound with a gene by engaging in interaction on the public channel tr. Here it receives two private channels *gene* and b that allows it to coordinate specifically with the gene in question.

Elongation The recursive (sub-)process, $\mathsf{rec}\,\mathrm{X}.\cdots$, either handles the next nucleotide in two steps:

- First, the type of nucleotide is communicated from gene to polymerase via the private channel *gene*.
- Then, if too long time passes before a corresponding NTP turns up, the polymerase breaks the coordination compound by sending a signal on the private channel b and then recurs to its initial state, ($\mathsf{rec}\,\mathrm{P}.\,\cdots$). Otherwise, the polymerase reacts with the NTP and recurs to handle the next nucleotide ($\mathsf{rec}\,\mathrm{X}.\,\cdots$).

Termination Or the genetic sequence (ACAC in the concrete case) is fully transcribed and the coordination compound is broken by a signal on b. The polymerase then recurs to its initial state ($\mathsf{rec}\,\mathrm{P}.\,\cdots$) while, at the same time, releasing a piece of messenger RNA corresponding to the transcribed sequence.

Note that this model does not pay any attention to the cooperativity issues of activators, inhibitors, genes, and polymerases. These issues, that are particularly relevant to the initiation phase, have been treated in great detail by Kuttler and Niehren [KN06] and also by Blossey, Cardelli, and Phillips [BCP06].

Also note that the model rests on the assumption that transcription may terminate prematurely. This type of behaviour, however, does not seem to be frequent in living systems; rather it seems to be connected with certain regulatory mechanisms that we shall not treat in detail. For our purposes the assumption simply results in a model that is slightly more complex and, thus, constitutes a more interesting subject of analysis in later chapters.

3.5 Concluding Remarks

In this chapter we have introduced a variant of the BioAmbients calculus [Reg03, RPS$^+$04] that incorporates general recursion in the manner of CCS [Mil80] and omits unrestricted choice in favour of guarded sum. The fundamental concepts of the calculus were introduced in three stages: First the fragment of *simple reactive processes*, which is strongly related to Milner's CCS [Mil80]. Then the fragment of *complex forming processes*, which is strongly related to the π-calculus by Milner, Walker, and Parrow [MPW92, Mil99]. And, finally, *compartment forming processes* in the form of the full BioAmbients calculus, which adds elements of Cardelli and Gordon's Mobile Ambients [CG00], but in the style of Levi and Sangiorgi's Mobile Safe Ambients [LS03] where all progress is a result of binary reactions rather than unary actions. One may observe that the resulting three classes of process expression correspond to the three classes of biological abstract machines identified by Cardelli [Car05a]:

The protein machine is concerned with *Biochemical Networks*, and seems to be adequately modelled by the notion of simple reactive processes. This is the view taken by Calder et al, who use the Performance Evaluation Process Algebra (PEPA), which is basically a stochastic extension of simple reactive processes, for the modelling and analysis of various signalling pathways [CGH06, CGH05, CDGH06, CGHV06], and, especially, Danos and Krivine in their work on formal biology in CCS [DK03]. Danos et al have made a few attempts at capturing special properties of the biological domain by addressing them at a more fundamental level: Reversible CCS aims to capture the concept of truly reversible reactions [DK03, DK04]. The κ-calculus focuses on the notion of complexation and its effect on the folding of proteins, in terms of exposure or hiding of binding sites, [DL03a, DL03b, DL04].

The Gene Machine is concerned with *Gene Regulatory Networks*, and seems to be adequately modelled by the notion of complex forming processes. This view was pioneered by Regev et al [RSS01], who worked with Priami [PRSS01] in order to base their BioSPI stochastic simulation tool on Priami's Stochastic version of the π-calculus [Pri96] coupled with Gillespie type rate estimations [Gil77, Gil76]. Technically this line of work has prospered at the hands of Phillips and Cardelli who have worked on graphical notations [PC05], simulation tools

[PC04], and modelling methodologies (Blossey et al) [BCP06]. The modelling efforts of Kuttler et al [KN06] have also led to technical extensions in the form of *concurrent objects* [Kut07]. Priami and Quaglia et al have fashioned *β-binders* [PQ04] – a calculus that combines features from the κ-calculus [DL04], the stochastic π-calculus [Pri96], and the ambient calculus [CG00]. in a manner that goes some way towards relinquishing the deterministic key-lock assumption in favour of a more nuanced notion of affinity.

The Membrane Machine is concerned with *Transport Networks*, and seems to be adequately modelled by the notion of membrane forming (BioAmbients) processes – again a view that was pioneered by Regev et al [Reg03, RPS$^+$04]. Other alternatives have emerged. Most notably Cardelli's *Brane calculus* [Car05b] that quite accurately models biomembranes as oriented 2-dimensional surfaces of bubbles that fuse and split. This model was simplified by Danos et al [DP04], but features for the modelling of large molecules and complexes have only been proposed quite recently [CP05]. A recent comparative analysis of BioAmbients and Brane calculus concludes that, from a meta point of view the two calculi are very similar [Ver07]. The β-binders [PQ04] also introduces a notion of boxed enclosures. These enclosures, however, do not nest and move, which renders the modelling of compartments less intuitive.

Chapter 4

Static Analysis Techniques

"It is the mark of an educated mind to rest satisfied with the degree of precision which the nature of the subject admits, and not to seek exactness where only an approximation is possible."
— Aristotle

Most semantic properties of Turing complete languages, e.g., BioAmbients, that can express any computable function, are *undecidable*, i.e., not generally computable within finite time and memory. This is a consequence of Rice's Theorem [Ric53], and the most well known example is Turing's famous *halting problem* [Tur36].

Small-step operational semantics [Plo04] expresses the behaviour of programs in terms of transition systems. In case the behaviour is finite we may compute the corresponding transition system by exhaustive simulation and subsequently check it for interesting properties. This is the approach of finite state *Model Checking* [CGP00], which often works well for state spaces of moderate size, but becomes intractable for large state spaces and undecidable for infinite ones.

In contrast, the basic tenet of *Static Analysis* [NNH99] is that a loss in precision often makes a property decidable, even for programs of infinite behaviour. Rather than computing the exact behaviour of a program, as defined by the semantics, a static analysis aims to compute an acceptable approximation to the behaviour directly from the static program code.

Traditionally, this type of analysis has been developed and used in the construction of *optimising compilers* [ASU86], where static analysis estimates serve as the basis of semantically safe *program transformations*. The design principles

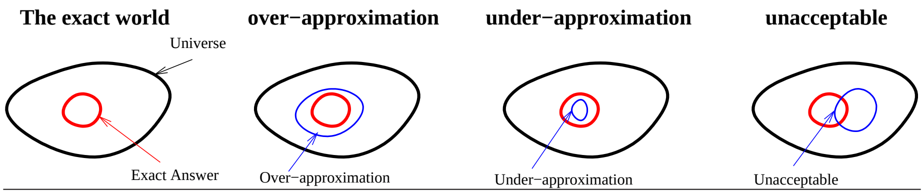


Figure 4.1: The nature of approximation.

that give static analyses their strength all owe to this heritage: **Exhaustive** – A static analysis should apply to all programs. Infinite behaviour must be acceptable. **Correct** – A static analysis should be safe with respect to the semantics. As shown in (Fig. 4.1), approximations are safe only if strict. **Implementable** – A static analysis should admit efficient implementations. Static analyses are meant to be invoked often. **Useful** – A static analysis should be useful. This generally requires a sensible compromise between the precision of the analysis and its decidability and computational tractability.

It is customary to distinguish between static Data Flow Analysis [Kil72], where one tracks how data moves through a collection of atomic computations, and Control Flow Analysis [Shi88], where one tracks how the point of control traverses a program. Traditionally, data flow properties have been the primary study in the classic setting of imperative languages, and control flow properties the primary study in the setting of functional languages. In process calculi, however, it is difficult to make a clear distinction between data and control structures. Thus, while it is often beneficial to approach process calculi with control flow techniques [BDNN98, HJNN99, NNH02, NNPdR07, NNB04], we shall be applying both control and data flow techniques to BioAmbients.

Classically there are two approaches to the design of static analyses. On the one hand, the design is *semantics directed* if the specification is calculated from a semantic specification; this is the approach of *Abstract Interpretation* [CC77, CC79]. On the other hand, the design is *semantics based* if the analysis information can be proved correct with respect to a semantic specification; this is the approach of *Monotone Frameworks* [KU77], *Flow Logic* [NN02], and *Type Systems* [Mil78]. As we shall see, the analyses specified in this dissertation are all semantics based.

The chapter is in four sections. We start in Section 4.1 by briefly reviewing basic order theory. Complete lattices, monotone functions, fixed points, and Moore families are all key notions in static analysis. In Section 4.2 we go on to introduce the central concepts of *Monotone Frameworks*. This is a classical concept in static analysis, which originally emerged as a generalisation of *Data Flow Analysis*. In Section 4.3 we turn to the main concepts of *Flow Logic*. This is a relative modern concept in static analysis, which originally emerged as a generalisation of *Control Flow Analysis*. Finally, in Section 4.4, we make some concluding remarks.

4.1 Order Theoretic Preliminaries

In preparation for the analysis sections we now give a cursory introduction to the notion of complete lattices, For a very detailed treatment of the subject, see [DP02] or perhaps [Tay99], and for a more succinct introduction, see [NNH99].

Definition 4.1 (Partial order) A *partial order* $(S, \sqsubseteq)$ is a set S accompanied by a binary relation $\sqsubseteq$ which is:

1. Reflexive: $\forall s \in S : s \sqsubseteq s$
2. Transitive: $\forall s, s', s'' \in S : s \sqsubseteq s' \wedge s' \sqsubseteq s'' \Rightarrow s \sqsubseteq s''$
3. Antisymmetric: $\forall s, s' \in S : s \sqsubseteq s' \wedge s' \sqsubseteq s \Rightarrow s = s'$

Notation 4.2 When necessary, we shall use subscripts to disambiguate the denotation of operators and e.g. write $\sqsubseteq_S$ rather than just $\sqsubseteq$. ■

If S has an element $s \in S$ such that $\forall s' \in S : s' \sqsupseteq s$ then this element is called the *least element* of S and is denoted $\bot$. Analogously, the *greatest element* of S is an element $s \in S$ such that $\forall s' \in S : s' \sqsubseteq s$ and is denoted $\top$. Generalising this leads to the definition of upper bounds:

Definition 4.3 (Upper bound) For a partial order $(S, \sqsubseteq)$ and subset $Y \subseteq S$ the element $s \in S$ is an *upper bound* of Y iff

$$\forall s' \in Y : s' \sqsubseteq s$$

Definition 4.4 (Least upper bound) For a partial order $(S, \sqsubseteq)$ and subset $Y \subseteq S$ the element s is a *least upper bound* (*lub*) of Y iff

1. s is an upper bound of Y, and
2. for every upper bound s' of Y, $s \sqsubseteq s'$.

Whenever $Y \subseteq S$ has a lub we denote it by $\bigsqcup Y$. ■

The binary least upper bound of $s, s' \in S$ is written $s \sqcup s'$.

The converse notions of a *lower bound* and a *greatest lower bound* (*glb*) may be defined similarly, and we write $s \sqcap s'$ for the binary glb of $s, s' \in S$.

Definition 4.5 (Lattice) Any non-empty partial order $(L, \sqsubseteq)$ that has $l \sqcup l'$ and $l \sqcap l'$ for all $l, l' \in L$, or equivalently $\bigsqcup Y$ and $\bigsqcap Y$ for all *finite* $Y \subseteq L$, is a *lattice*.

Definition 4.6 (Complete lattice) Any non-empty partial order $(L, \sqsubseteq)$ that has $\bigsqcup Y$ and $\bigsqcap Y$ for all $Y \subseteq L$ is a *complete lattice*. ■

Note that, if $(L, \sqsubseteq)$ is a complete lattice, then $\bot = \bigsqcup \emptyset = \bigsqcap L$ is the *least element* and $\top = \bigsqcap \emptyset = \bigsqcup L$ is the *greatest element*. Furthermore, Definitions 4.5 and 4.6 ensure that any finite lattice is complete.

Proposition 4.7 (Cartesian product) If $(L_1, \sqsubseteq_{L_1})$ and $(L_2, \sqsubseteq_{L_2})$ are both complete lattices then so is the *Cartesian product*

$$(L_1 \times L_2, \sqsubseteq_{L_1 \times L_2})$$

where the *componentwise order* is defined by

$$(l_1, l_2) \sqsubseteq_{L_1 \times L_2} (l'_1, l'_2) \text{ iff } l_1 \sqsubseteq_{L_1} l'_1 \wedge l_2 \sqsubseteq_{L_2} l'_2$$

Proof See e.g. [NNH99]. □

Proposition 4.8 (Function space) If S is any set and $(L, \sqsubseteq_L)$ is a *complete lattice* then so is the *function space*

$$(S \to L, \sqsubseteq_{S \to L})$$

where the *pointwise order* is defined by

$$f \sqsubseteq_{S \to L} g \text{ iff } \forall s \in S : f(s) \sqsubseteq_L g(s)$$

Proof See, e.g., [NNH99]. □

Definition 4.9 (Moore family) A subset, Y, of a complete lattice, $\subseteq (L, \sqsubseteq)$, is a *Moore family* if it is closed under greatest lower bounds, i.e.,

$$\forall Y' \subseteq Y : \bigsqcap Y' \in Y$$

■

Note that a Moore family always contains a least element, $\bigsqcap Y$, and a greatest element, $\bigsqcap \emptyset = \top_L$; thus it is never empty.

Definition 4.10 (Monotone function) A function, $f : S_1 \to S_2$, between partially ordered sets, $(S_1, \sqsubseteq_{S_1})$ and $(S_2, \sqsubseteq_{S_2})$, is *monotone* if

$$\forall s, s' : s \sqsubseteq_{S_1} s' \Rightarrow f(s) \sqsubseteq_{S_2} f(s')$$

■

Definition 4.11 (Chain) A subset, Y, of a partial order, $(S, \sqsubseteq)$, is a *chain* if it is totally ordered, i.e.,

$$\forall s, s' \in Y : (s \sqsubseteq s') \vee (s' \sqsubseteq s)$$

If Y is a finite subset then it constitutes a *finite chain*.

Definition 4.12 (Ascending chain condition) A partial order, $(S, \sqsubseteq)$, satisfies the *ascending chain condition* if any ascending chain, $\{s_1, s_2, \ldots, s_n, \ldots\} \subseteq S$, for which $s_1 \sqsubseteq s_2 \sqsubseteq \cdots \sqsubseteq s_n \sqsubseteq \cdots$ eventually stabilises, i.e.,

$$\exists k \in \mathbb{N} : s_k = s_{k+1} = \ldots$$

■

Note, that this condition is trivially satisfied by any finite partial order.

Definition 4.13 (Fixed point) Consider a monotone function, $f : L \to L$, on a complete lattice, $(L, \sqsubseteq)$. A *fixed point* of f is an element, $l \in L$, such that $f(l) = l$. We write

$$\text{Fix}(f) = \{l \mid f(l) = l\}$$

for the set of such fixed points. ■

Proposition 4.14 (Least fixed point) Any monotone function, $f : L \to L$, on a complete lattice, $(L, \sqsubseteq)$, has a unique least fixed point, LFP(f), defined by:

$$\text{LFP}(f) = \textstyle\bigsqcup_{i \geq 0} f^i(\bot)$$

for which $f(\text{LFP}(f)) = \text{LFP}(f)$.

Proof See, e.g., [NNH99]. □

Clearly, if L satisfies the ascending chain condition, then LFP(f) can be computed in a finite number of iterations.

4.2 Monotone Frameworks

Data Flow Analysis problems emerge when one tries to compute how data moves through a collection of atomic computations. Monotone Frameworks constitute a generalised approach to solving such problems. Originally invented as the basis of safe transformations within optimising compilers they date back to the work of Kam and Ullman [KU77], who generalised Kildall's lattice theoretic approach to Data Flow Analysis [Kil72]. For a thorough introduction to the subject see Aho, Sethi, and Ullman [ASU86] or Nielson, Nielson, and Hankin [NNH99].

Basic Concepts. Originally the notion of global Data Flow Analysis problems arose in the context of imperative languages. In this context it is natural to think of programs, P, as *flow-graphs* where the nodes denote so-called *elementary blocks* and the edges denote *flow of control*. Generally, the notion of elementary blocks depends on the language at hand. However, we shall think of them simply as the least pieces of code that are of interest to the analysis problem and has one entry point, one exit point, and no internal looping.

Example 4.15 Consider the following minimal imperative language:

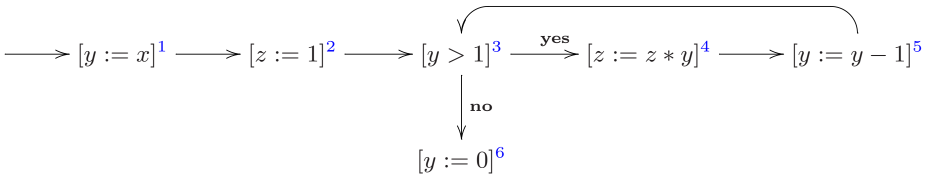
x	$\in$	$\mathbf{V}$	variables
n	$\in$	$\mathbf{C}$	constants
ℓ	$\in$	$\mathbf{Lab}$	labels

a	$::=$	$x \mid n \mid \cdots$	**Arithmetic expressions**
b	$::=$	$t \mid f \mid \cdots$	**Boolean expressions**
S	$::=$		**Program Statements**
		$[x{:=}a]^\ell$	assignment statement
	$\mid$	$S_1;\ S_2$	sequential statement
	$\mid$	$\mathsf{while}\ [b]^\ell\ \mathsf{do}\ S$	loop statement

where the labelled entities $[B]^\ell$ are the elementary blocks. In this context the factorial program

$$S_{\mathsf{fact}} = [y := x]^1; [z := 1]^2; \mathsf{while}\ [y > 1]^3\ \mathsf{do}\ ([z := z * y]^4; [y := y - 1]^5); [y := 0]^6$$

may be thought of as the flow-graph



$$\begin{tikzcd}
{} \arrow[r] & {[y := x]^1} \arrow[r] & {[z := 1]^2} \arrow[r] & {[y > 1]^3} \arrow[r, "\text{yes}"] \arrow[d, "\text{no}"] & {[z := z * y]^4} \arrow[r] & {[y := y - 1]^5} \arrow[lll, bend right] \\
 & & & {[y := 0]^6} & &
\end{tikzcd}$$

$\square$

The aim of a Monotone Framework is to approximate some interesting property of the data flow of a given program. For this purpose it is useful to think of the program as a transition system. The states correspond to the data flow information available at the entry points of elementary blocks. The transitions correspond to the effects of the elementary blocks on the information. In this setting it is the information of the states that we seek to approximate.

Example 4.16 Consider S_{fact} of Example 4.15. In terms of data flow this program might be seen as a non-deterministic transition system

$$\begin{tikzcd}
q_1 \arrow[r, "1"] & q_2 \arrow[r, "2"] & q_3 \arrow[r, "3"] \arrow[d, "3"] & q_4 \arrow[r, "4"] & q_5 \arrow[ll, bend right, "5"'] \\
 & & q_6 \arrow[r, "6"'] & \circledcirc &
\end{tikzcd}$$

where q_1 is the initial state and each edge corresponds to the effect of the elementary block of the same label. $\square$

Property Space. In order to reason about this transition system we need a representation of states. The corresponding universe of discourse is called a *property space.*

It is customary to demand that the property space is a complete lattice

$$(L, \sqsubseteq)$$

that satisfies the ascending chain condition. It should be designed such that states of the transition system of interest correspond to elements of L. Furthermore, the least upper bound operation $\bigsqcup$ associated with L must be suitable for safely combining the effects of transitions whenever multiple transitions enter the same state. The ascending chain condition helps to ensure that the Data Flow approximation may be computed in a finite manner.

Transfer Function Space. We also need a representation of the effect of transitions. The corresponding universe of discourse is called a *transfer function space.*

Here it is customary to demand that the transfer function space is a set of functions

$$F = \{f : L \to L \mid f \text{ is monotonic}\}$$

that contains the identity function id and is closed under composition. It should be designed such that the transitions of the transition system of interest correspond to elements of F. The monotonicity requirement helps to ensure that the Data Flow approximation may be computed in a finite manner.

Monotone Framework. These two ingredients formalise a given Data Flow Analysis problem as a *Monotone Framework.*

It is normal to refer to the Monotone Framework of some global Data Flow Analysis problem, A, as a triple

$$\mathsf{A} = (L, \bigsqcup_L, F)$$

consisting of a suitable property space, L, the corresponding least upper bound operation, $\bigsqcup_L$, and a suitable transfer function space, F.

If all functions f in F are required to be *distributive*, i.e., $f(l_1 \sqcup l_2) = f(l_1) \sqcup f(l_2)$, the framework specialises to a *distributive framework.* This concept is somewhat stronger as it often allows for more efficient algorithms [NNH99].

If furthermore the property space is restricted to

$$L = (\mathcal{P}(D), \sqsubseteq)$$

for some finite set D and $\sqsubseteq = \subseteq$ or $\sqsubseteq = \supseteq$, then the transfer function space can invariably be restricted to

$$F = \{f : L \to L \mid \exists l_k, l_g \in L : f(l) = (l \setminus l_k) \cup l_g\}$$

and the framework further specialises to a *bitvector framework.*

Example 4.17 Consider the global Data Flow Analysis problem of *Reaching Definitions*:

RD: What definitions (assignments) may reach what basic blocks?

The goal of the corresponding analysis is to determine, for every program point, the origin of every variable assignment that might be in effect when the point is reached.

The property space of interest, L_{RD}, is the set of functions from finite subsets of $\mathbf{V}$ to $\mathbf{Lab}^?$, where $\mathbf{Lab}^?$ is the infinite set of labels $\mathbf{Lab}$ extended with $?$ (meaning 'undefined'), i.e. $L_{\mathsf{RD}} \subset \mathcal{P}(\mathbf{V} \times \mathbf{Lab}^?)$. This is a complete lattice where $\bigsqcup = \bigcup$, $\sqsubseteq = \subseteq$, $\bot = \emptyset$.

In order to define the corresponding transfer function space we first define two helpful functions. The generate function

$$\mathsf{gen}_{\mathsf{RD}}[x := a]^\ell = \{(x, \ell)\}$$
$$\mathsf{gen}_{\mathsf{RD}}[b]^\ell = \emptyset$$

defines how elementary blocks contribute new information, and the kill function

$$\begin{aligned}\mathsf{kill}_{\mathsf{RD}}[x := a]^{\ell} &= \{(x, ?)\} \cup \{(x, \ell') \mid (x, \ell') \in l\} \\ \mathsf{kill}_{\mathsf{RD}}[b]^{\ell} &= \emptyset\end{aligned}$$

defines how elementary blocks invalidate old information. Now, let $l \in L$. Then

1. if $x \in \mathbf{V}$ and $\ell \in \mathbf{Lab}$ then $(\lambda l.(l \setminus \mathsf{kill}_{\mathsf{RD}}[x := a]^{\ell}) \cup \mathsf{gen}_{\mathsf{RD}}[x := a]^{\ell}) \in F_{\mathsf{RD}}$,
2. if $\ell \in \mathbf{Lab}$ then $(\lambda l.(l \setminus \mathsf{kill}_{\mathsf{RD}}[b]^{\ell}) \cup \mathsf{gen}_{\mathsf{RD}}[b]^{\ell}) \in F_{\mathsf{RD}}$,
3. $(\lambda l.l) \in F_{\mathsf{RD}}$, and
4. if $f_1, f_2 \in F$ then $(f_1 \circ f_2) \in F_{\mathsf{RD}}$.

□

Instances. When a general Monotone Framework is confronted with a particular program, $P_\star$, of interest, a Data Flow Analysis emerges as a concrete instance of the framework.

Such an instance comprises a number of elements:

- The property space $(L, \bigsqcup)$
- The transfer function space F
- A finite transition system $(\mathsf{Q}, \mathsf{q}_\star, \delta)$ where Q is the set of program points in $P_\star$. δ is a labelled transition relation representing the flow-graph of $P_\star$, and $\mathsf{q}_\star$ is the (set of) distinguished entry point(s).
- An extremal value ι, to be associated with the entry point(s).
- A mapping $f_{\_} : \mathbf{Lab}_\star \to F$

where $\mathbf{Lab}_\star$ is the set of labels actually occuring in $P_\star$.

It also gives rise to a set of constraints, $\mathsf{A}^{\sqsubseteq}$, defined by:

$$\mathsf{A}[q_t] \sqsupseteq \bigsqcup\{f_{\ell}(\mathsf{A}[q_s]) \mid (q_s, \ell, q_t) \in \delta\} \sqcup \iota_{\mathsf{q}_\star}^{q_t}$$

where

$$\iota_{\mathsf{q}_\star}^{q_t} = \begin{cases} \iota & \text{if } q_t \in \mathsf{q}_\star \\ \bot & \textit{otherwise} \end{cases}$$

Example 4.18 Consider S_{fact} of Example 4.15. It yields the following instance of RD:

$$
\begin{aligned}
L &= \mathcal{P}(\mathbf{V}_{\mathsf{fact}} \times \mathbf{Lab}^{?}_{\mathsf{fact}}) \subset \mathcal{P}(\mathbf{V} \times \mathbf{Lab}^{?}) \\
F & \text{ is as } F_{\mathsf{RD}} \text{ but restricted to } \mathbf{V}_{\mathsf{fact}} \text{ and } \mathbf{Lab}_{\mathsf{fact}} \\
\mathsf{q}_\star &= \{q_1\} \\
Q &= \{q_1, \cdots, q_6\} \\
\delta &= \{(q_1, 1, q_2), (q_2, 2, q_3), (q_3, 3, q_4), (q_3, 3, q_6), (q_4, 4, q_5), (q_5, 5, q_3)\} \\
f_\ell(l) &= (l \setminus \mathsf{kill}_l[B]^\ell) \cup \mathsf{gen}_l[B]^\ell \text{ where } [B]^\ell \text{ is an elementary block of } S_{\mathsf{fact}} \\
\iota &= \{(x, ?) \mid x \in \mathbf{V}_{\mathsf{fact}}\} \\
\mathsf{RD}^{\sqsubseteq} &= \{\iota \sqsubseteq \mathsf{RD}[q_1],\ (f_2(\mathsf{RD}[q_2]) \sqcup f_5(\mathsf{RD}[q_5])) \sqsubseteq \mathsf{RD}[q_3], \\
&\qquad f_1(\mathsf{RD}[q_1]) \sqsubseteq \mathsf{RD}[q_2],\ f_3(\mathsf{RD}[q_3]) \sqsubseteq \mathsf{RD}[q_4], \\
&\qquad f_3(\mathsf{RD}[q_3]) \sqsubseteq \mathsf{RD}[q_6],\ f_4(\mathsf{RD}[q_4]) \sqsubseteq \mathsf{RD}[q_5]\}
\end{aligned}
$$

Note that $\mathcal{P}(\mathbf{V}_{\mathsf{fact}} \times \mathbf{Lab}^{?}_{\mathsf{fact}})$ is finite, in contrast to $\mathcal{P}(\mathbf{V} \times \mathbf{Lab}^{?})$, and hence the instance satisfies the ascending chain condition. □

Remark 4.19 Note, that in the context of an imperative language the transition system $(\mathsf{Q}, \mathsf{q}_\star, \delta)$ is a priori fixed by the flow-graph, and only a solution, $\mathcal{A}_\star \in (\mathbf{Lab}_\star \to L)$, to $\mathsf{A}^{\sqsubseteq}$remains to be computed. As we shall see in Chapter 8, this is not necessarily the case for process calculi. ■

Worklist Algorithm. The constraint system, $\mathsf{A}^{\sqsubseteq}(P)$, derived from P constitutes a declarative specification of the solution space. We are interested in the most informative solution, $\mathsf{A}_\star$, which emerges as the least fixed point of the constraint system. In the terminology of Monotone Frameworks this is called the Meet Over all Paths (MOP) and is sometimes hard to compute. In the case of Distributive Frameworks, however, it coincides with the so-called Maximal Fixed Point (MFP), which is computed by the worklist algorithm shown in Table 4.1.

Example 4.20 For S_{fact} the following result emerges as the MFP solution to $\mathsf{RD}^{\sqsubseteq}$, and we say that $\mathsf{RD}_{\mathsf{fact}} \models \mathsf{RD}^{\sqsubseteq}(S_{\mathsf{fact}})$:

$$
\begin{aligned}
\mathsf{RD}_{\mathsf{fact}}[q_1] &= \{(x, ?), (y, ?), (z, ?)\} \\
\mathsf{RD}_{\mathsf{fact}}[q_2] &= \{(x, ?), (y, 1), (z, ?)\} \\
\mathsf{RD}_{\mathsf{fact}}[q_3] &= \{(x, ?), (y, 1), (z, 2), (y, 5), (z, 4)\} \\
\mathsf{RD}_{\mathsf{fact}}[q_4] &= \{(x, ?), (y, 1), (z, 2), (y, 5), (z, 4)\} \\
\mathsf{RD}_{\mathsf{fact}}[q_5] &= \{(x, ?), (y, 1), (y, 5), (z, 4)\} \\
\mathsf{RD}_{\mathsf{fact}}[q_6] &= \{(x, ?), (y, 1), (z, 2), (y, 5), (z, 4)\}
\end{aligned}
$$

□

```
INPUT  : Instance $((L, F), (\mathsf{Q}, \mathsf{q}_\star, \delta), \iota, f_{\_})$
OUTPUT : Least solution $\mathsf{A}_\star$ such that $\mathsf{A}_\star \models \mathsf{A}^{\sqsubseteq}$
METHOD : Step 1:  Initialisation of $\mathsf{W}$ and $\mathsf{A}$
                  $\mathsf{W} = \mathsf{q}_\star$;
                  for all $q$ in $\mathsf{Q}$ do
                      if $q \in \mathsf{q}_\star$
                      then $\mathsf{A}[q] := \iota$
                      else $\mathsf{A}[q] := \bot$
         Step 2:  Iteration (updating $\mathsf{W}$ and $\mathsf{A}$)
                  while $\mathsf{W} \neq \emptyset$ do
                      select $q_s$ from $\mathsf{W}$;  $\mathsf{W} := \mathsf{W} \setminus \{q_s\}$;
                      for each $(q_s, \ell, q_t)$ in $\delta$ do
                          let $l := f_\ell(\mathsf{A}[q_s])$ in
                              if $l \not\sqsubseteq \mathsf{A}[q_t]$
                              then $(\mathsf{A}[q_t] := \mathsf{A}[q_t] \sqcup l$;  $\mathsf{W} := \mathsf{W} \cup \{q_t\})$
         Step 3:  Presenting the result
                  $\mathsf{A}_\star := \mathsf{A}$
```

Table 4.1: Maximal Fixed Point algorithm for Monotone Frameworks.

Desirable Properties. A number of properties are desirable for a Monotone Framework; hence their presence is subject to proof.

Termination. The analysis must be exhaustive and compute a result for all programs. We prove this property by showing that the worklist algorithm always terminates. Here, this is ensured by the ascending chain property and monotonicity, but in Chapter 8, where we define the so-called Pathway Analysis, we shall have to prove termination.

Preservation of Solutions. The solutions produced by the worklist algorithm must be preserved under the semantics. This can be proved by showing a result of the following kind:

$$\text{If } \mathsf{A} \models \mathsf{A}^{\sqsubseteq}(P) \text{ and } P \longrightarrow^{\star} Q \text{ then } \mathsf{A} \models \mathsf{A}^{\sqsubseteq}(Q)$$

Intuitively, this expresses that analysis results remain valid as execution commences, and, as we shall see, this is one of the most fundamental properties of a static analysis.

Correctness. The solution, however, must also correctly capture the property of interest.

This might be a property related to, e.g., the memory state of semantic configurations $\langle P, s\rangle$, in the manner of structural operational semantics [Plo04]. In this case it is customary to define relation $\mathfrak{R} \subseteq (\mathbf{Lab} \times L) \times \mathbf{State}$ that formally captures the notion of correctness, i.e.,

$$\text{If } \mathsf{A} \models \mathsf{A}^{\sqsubseteq}(P),\ \mathsf{A}\,\mathfrak{R}\,s,\ \text{and } \langle P, s\rangle \longrightarrow^{\star} \langle Q, s'\rangle \text{ then } \mathsf{A}\,\mathfrak{R}\,s'$$

Alternatively, as is more common for process calculi equipped with a reduction style semantics, it can be a property related to the sequence of events that lead to a certain configuration P. In this case one often relies on a collecting semantics, i.e., an instrumented version that keeps track of additional information that pertains to the analysis. Thus, all configurations include a trace, $tr \in \mathbf{Trace}$, of such information, and we define a correctness relation, $\mathfrak{R} \subseteq (\mathbf{Lab} \times L) \times \mathbf{Trace}$. Correctness of the analysis then follows from:

$$\text{If } \mathsf{A} \models \mathsf{A}^{\sqsubseteq}(P),\ \mathsf{A}\,\mathfrak{R}\,tr,\ \text{and } \langle P, tr\rangle \longrightarrow^{\star} \langle Q, tr'\rangle \text{ then } \mathsf{A}\,\mathfrak{R}\,tr'$$

Remark 4.21 Note that the collecting semantics of BioAmbients collects such traces, in the sense that $\langle P_\star, \varepsilon\rangle$, and $\langle P_\star, \varepsilon\rangle \xrightarrow{\tilde{L}}{}^{\star} \langle Q, \tilde{L}\rangle$. We shall use this later when formulating our correctness results. ∎

4.3 Flow Logic

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 [Shi88], and was further refined by Jagannathan [JW95]. In this context, and based on the view that, in terms of flow, data and control are two sides of the same thing [NNH99], *Flow Logic* was pioneered in the late 1990's, by Nielson and Nielson [NN98, NN97, NN02] as a unifying specification oriented approach to constraint based static analysis.

Basic Concepts. 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 implementor is always free to choose the best available tool — no particular tool or formalism is prescribed by the framework.

Analysis Domain. As for Monotone Frameworks, a Flow Logic specification is based on a suitable universe of discourse, L. Here it is customary to follow the approach of Monotone Frameworks and demand that L is a complete lattice. However, the domain of a Flow Logic specification should be designed such that elements correspond to *global* Control Flow Analysis information of the program of interest. This is in contrast to the property space of a monotone framework, which is designed such that the elements correspond to the Data Flow Analysis information of individual program points (or states if you will) of the program of interest,

Acceptability Judgement. Fundamentally, a Flow Logic specification is concerned with the relationship between programs $P \in \textsc{lang}$ and static analysis estimates $\mathcal{A} \in L$. 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 $\textsc{lang}$ and they take the form

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

where φ is usually a formula in a suitable fragment of First Order Logic, $\textsc{fol}$.

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

$$\models : (L \times \textsc{lang}) \to \{\text{true}, \text{false}\}$$

Generally, as we do not demand the specification to be *syntax directed*, i.e., for each of the defining clauses that each P' occurring in φ is one of the P_i occurring in $\phi(\cdots P_i \cdots)$, we have to define the meaning of "$\models$" by *co-induction* rather than induction.

Thus, when formally assigning meaning to a Flow Logic specification, we do so by regarding it as defining a functional

$$\mathcal{Q} : ((L \times \textsc{lang}) \to \{\text{true}, \text{false}\}) \to ((L \times \textsc{lang}) \to \{\text{true}, \text{false}\})$$

the *greatest fixed point* of which we take as the formal meaning.

However, when the specification *is* syntax directed the least and greatest fixed points coincide, and hence "$\models$" is adequately defined by ordinary induction.

In the terminology of Flow Logic, and by analogy to Structural Operational Semantics [Plo04], we call such specifications *compositional*. Specifications that are not syntax directed we call *abstract*.

Desirable Properties. For a setup like this to qualify as a Flow Logic it must posses a number of desirable properties that we shall elaborate in the following:

Well-definedness. The analysis must be well-defined, i.e., for every combination of $P \in \textsc{lang}$ and $\mathcal{A} \in L$, 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 the case of a compositional specification this is immediate to show by structural induction in P. In the case of an abstract specification, however, the judgement is well-defined only if $\mathcal{Q}$ constitutes a monotone functional over the complete lattice

$$((L \times \textsc{lang}) \rightarrow \{\text{true}, \text{false}\}, \sqsubseteq)$$

where the ordering $\sqsubseteq$ is given by:

$$Q_1 \sqsubseteq Q_2 \text{ iff } \forall(\mathcal{A}, P) : (Q_1(\mathcal{A}, P) = \text{true}) \Rightarrow (Q_2(\mathcal{A}, P) = \text{true})$$

When this is the case the existence of a suitable greatest fixed point is guaranteed by Tarski's fixed point theorem.

Semantic Correctness. Intuitively, an analysis is *semantically correct* if analysis estimates that are acceptable for a process P contain information about every possible evolution of P that is allowed by the semantics. As Flow Logic is a semantics based approach to static analysis this notion of correctness is usually captured by a *subject reduction* result:

$$\text{if } \mathcal{A} \models P \text{ and } P \longrightarrow Q \text{ then } \mathcal{A} \models Q$$

which expresses that the acceptability of analysis estimates is preserved by the reaction relation. That is, if $\mathcal{A}$ is an acceptable estimate for P, and the semantics allows P to evolve into Q, then $\mathcal{A}$ is also acceptable for Q. This relies on an auxiliary result for structural congruence:

$$\text{if } P \equiv Q \text{ then } \mathcal{A} \models P \text{ iff } \mathcal{A} \models Q$$

asserting that the acceptability of analysis estimates is invariant under the structural congruence.

Remark 4.22 Due to the directedness of $\Rrightarrow$ the latter becomes

$$\text{if } \mathcal{A} \models P \text{ and } P \Rrightarrow Q \text{ then } \mathcal{A} \models Q$$

in our setup. ■

Clearly, full semantic correctness, sometimes called *semantic soundness* follows by the transitive closure of the subject reduction result, i.e.,

$$\text{if } \mathcal{A} \models P \text{ and } P \longrightarrow^{\star} Q \text{ then } \mathcal{A} \models Q$$

Occasionally, it is convenient to use a collecting semantics and formalise the relationship, $\mathfrak{R} \subseteq L \times \textbf{Trace}$, between analysis estimates and traces in order to formulate correctness:

$$\text{if } \mathcal{A} \models P, \mathcal{A}\,\mathfrak{R}\,tr, \text{ and } \langle P, tr\rangle \longrightarrow^{\star} \langle Q, tr'\rangle, \text{ then } \mathcal{A} \models Q \text{ and } \mathcal{A}\,\mathfrak{R}\,tr'$$

Moore Family Property. 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:

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

Recall that, trivially, a Moore family is never empty as it always contains a greatest element $\bigsqcap \emptyset = \top_L$, which is the trivial *worst* (i.e., least informative) acceptable analysis estimate. In contrast it is also guaranteed to have a least element $\mathcal{A}_\star = \bigsqcap\{\mathcal{A} \mid \mathcal{A} \models P\}$, which is the least admissible result under the ordering $\sqsubseteq_L$ of L and, hence, the *best* (most informative) acceptable estimate.

Implementation. The implementation of a Flow Logic specification is enabled by a simple change of viewpoint: Intuitively, the acceptability judgement

$$\mathcal{A} \models P \quad \text{iff} \quad \varphi$$

associates each program, P, with a formula, φ, such that an analysis estimate, $\mathcal{A}$, is acceptable for P if, and only if, $\mathcal{A}$ constitutes a model of φ. Thus, as long as we are able to compute the appropriate φ the remaining task of finding a suitable model, $\mathcal{A}$, can be left to an auxiliary logical solver.

In the presence of an abstract Flow Logic specification this idea clearly breaks down because φ is not necessarily finite for every given P. In the case of a compositional Flow Logic specification, however, the finiteness of φ follows directly from the finiteness of the syntactic representation of P and the inductive nature of the specification. If, furthermore, the specification is well-defined then it defines a total function from programs $P \in \textsc{lang}$ to formulae $\varphi \in \textsc{fol}$. And when this is the case we shall refer to the Flow Logic specification of some global Control Flow Analysis problem, A, as a binary predicate

$$\mathsf{A} = \mathcal{FL}(\textsc{lang}, \textsc{fol}).$$

The ALFP Logic. Clearly, the theoretical as well as practical properties of an

$$
\begin{array}{lcl}
\mathsf{term} & ::= & c \mid x \mid f(\mathsf{term}_1, \ldots, \mathsf{term}_k) \\
\\
\mathsf{pre} & ::= & R(\mathsf{term}_1, \ldots, \mathsf{term}_k) \mid \neg R(\mathsf{term}_1, \ldots, \mathsf{term}_k) \\
& \mid & \mathsf{term}_1 = \mathsf{term}_2 \mid \mathsf{term}_1 \neq \mathsf{term}_2 \\
& \mid & \mathsf{pre}_1 \wedge \mathsf{pre}_2 \mid \mathsf{pre}_1 \vee \mathsf{pre}_2 \mid \forall x : \mathsf{pre} \mid \exists x : \mathsf{pre} \\
\\
\mathsf{clause} & ::= & R(x_1, \ldots, x_k) \mid \mathbf{1} \mid \mathsf{clause}_1 \wedge \mathsf{clause}_2 \\
& \mid & \mathsf{pre} \Longrightarrow \mathsf{clause} \mid \forall x : \mathsf{clause}
\end{array}
$$

Table 4.2: Syntax of the Alternation-free Least Fixed Point Logic.

analysis implementation depend on the formulae, φ, derived from programs, P, as well as the solver used for the computation of models.

In order to ensure nice properties, the Flow Logic specifications presented in this dissertation are expressed using an extension of Horn clauses, known as Alternation-free Least Fixed Point Logic (ALFP) [NNS02]. This logic is presented in Table 4.2, where we write x for variables, c for constants, f for function symbols, R for predicates, term for terms, pre for preconditions, and clause for clauses.

The clauses are interpreted over a universe $\mathcal{U}$ of ground terms. The semantics is given in terms of satisfaction relations

$$(\rho, \sigma) \models \mathsf{pre} \text{ and } (\rho, \sigma) \models \mathsf{clause}$$

that are defined in the standard way. Thus, ρ is an interpretation of predicate symbols, σ is an interpretation of terms, and the clause, e.g., $R(x_1, \ldots, x_k$ is interpreted according to the following rule:

$$(\rho, \sigma) \models R(x_1, \ldots, x_k) \quad \text{iff} \quad (\sigma x_1, \ldots, \sigma x_2) \in \rho R$$

The logic has the nice feature that clauses with no free variables satisfy the *model intersection property*, i.e., given an interpretation σ_0 of the free variables in a clause cl, the set of interpretations $\{\rho \mid (\rho, \sigma_0) \models \mathsf{cl}\}$ constitutes a Moore family [NNS02].

Note that in the presence of negation in preconditions the solvability of clauses is subject to a notion of stratification, which is not relevant, however, in the context of this dissertation.

As a matter of convenience we shall often use other mathematical notation to write certain ALFP constructs:

```
INPUT :     a Flow Logic $\mathcal{FL}(\textsc{lang}, \textsc{alfp})$ and
            a program $P$.
OUTPUT :    an ALFP formula $\varphi$ such that $\mathcal{A} \models \varphi \Leftrightarrow \mathcal{A} \models P$.
METHOD :    Set $\varphi := \mathcal{A} \models P$
            while $\varphi$ contains $\mathcal{A} \models P'$
              and there is a rule $\alpha$ iff $\beta$ in $\mathcal{FL}$
                   and a substitution $\theta$
              such that $\theta\alpha = \mathcal{A} \models P'$
             do replace $\mathcal{A} \models P'$ with $\theta\beta$ in $\varphi$.
```

Table 4.3: Constraint generation algorithm for Flow Logics.

- The predicates of ALFP represent relations. For a such a relation R we shall often write $(x_1, \ldots, x_k) \in R$ to denote the ALFP atom $R(x_1, \ldots, x_k)$.

- When testing for intersection we often write $R(x_1, \ldots, x_i) \cap R(y_1, \ldots, y_i) \neq \emptyset$ to denote $\exists z_j, \ldots, z_k : R(x_1, \ldots, x_i, z_j, \ldots, z_k) \wedge R(y_1, \ldots, y_i, z_j, \ldots, z_k)$.

- When expressing subset relationships we write $R(x_1, \ldots, x_i) \subseteq R(y_1, \ldots, y_i)$ to denote $\forall z_j, \ldots, z_k : R(x_1, \ldots, x_i, z_j, \ldots, z_k) \Rightarrow R(y_1, \ldots, y_i, z_j, \ldots, z_k)$.

Clause Generation. Algorithmically we capture the required change of viewpoint in a clause generator that works largely as the chaotically iterative algorithm outlined in Table 4.3. This procedure takes as input a Flow Logic $\mathcal{FL}(\textsc{lang}, \textsc{alfp})$ and a program P. And, from this, it produces an ALFP clause, the models of which are exactly the acceptable analysis estimates.

The best (i.e., least under the ordering of L) acceptable analysis estimate $\mathcal{A}_\star$ for $P_\star$ is the minimal model satisfying the generated clause. The model intersection property of ALFP guarantees that this can be unambiguously computed using a fixed point engine like those of *The Succinct Solver Suite* [NNS$^+$04].

Remark 4.23 Generally analyses implemented using the succinct solver exhibit an asymptotic worst-case complexity that is of the order of $\mathcal{O}(n^{k+1})$, where n is the size of the term universe, which is fixed for any given process, and k is the maximal nesting depth of quantifiers in the generated clause.

4.4 Concluding Remarks

In this chapter we have introduced two common approaches to Static Program Analysis. These approaches originate from the analytic needs of two very different programming paradigms and thus differ in some respects.

Classically, Monotone Frameworks constitute a generalised approach to the Data Flow Analysis problems of imperative languages – a setting where programs are normally considered as flow-graphs. However, we have deliberately presented the approach in a setting where programs are considered as labelled transition systems. This sets the scene for Chapter 8, where we shall use the approach to approximate the temporal structure, i.e., the underlying transition systems defined by the semantics, of BioAmbients models.

Similarly, Flow Logic constitutes a generalised approach to the Control Flow Analysis problems of functional languages – a setting where continuations are data and the flow of control is predominantly decided at run-time, Here it has traditionally been used to approximate the control flow structures of programs in the context of dynamic dispatch. In Chapters 6 and 7 we shall similarly use Flow Logic to approximate the spatial structure, i.e., the set of ambient induced nesting hierarchies, that can arise dynamically from a BioAmbients model.

Throughout the technical developments we shall consistently take the approach that is inherent in semantics based program analysis: In order to study properties of a program $P_\star$ we analyse the static snapshot given by the direct syntactical representation of $P_\star$. However, we carefully define the analyses such that the properties are "$\sqsupseteq$-preserved" under heating and reaction and use this to establish the semantic soundness. In this manner we ensure that the information learnt from the study of $P_\star$ in $P_\star \xrightarrow{\tilde{L}}{}^\star Q$ is also valid for Q.

Part II

Analysing for Structural Properties

Chapter 5

Well-formed Programs and Their Properties

"Well-typed programs don't go wrong."

— Robin Milner

This chapter introduces a notion of well-formed programs defined by a static well-formedness condition in the manner of type systems [NNH99]. The presentation largely covers material that is also covered in [PNN07].

When defining the syntax and semantics of BioAmbients in Chapter 3, we deliberately postponed the formal definition of several central notions, such as the free names and identifiers of processes, identifier substitution, and name substitution. This we did because many of these notions are complicated by the presence of general recursion. In particular, it is important to enforce static scope if α-equivalence is included in the structural congruence (heating) relation [PV05]. There are three viable approaches to this:

One option is to define recursion via a notion of parametrised process constants

$$\mathsf{A}(\vec{p}) \triangleq P$$

where $\mathsf{fn}(P) \subseteq \{\vec{p}\}$ [Mil99]. However, this immediately turns seemingly simple properties, such as free names, into fixpoint properties [NNP07].

Another option is to define α-equivalence and α-renaming for all entities that risk capture, i.e., constants, variables, and process identifiers and then dynamically enforce static scope. However, this results in quite complicated notions of

$$\begin{tikzcd}
 & & \mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P \\
\mathsf{fn}_{\Gamma_{\mathsf{fn}}}(P) \arrow[urr] & \mathsf{fpi}(P) \arrow[ur] &
\end{tikzcd}$$

Figure 5.1: The hierarchy of notions that defines well-formedness.

substitution that rely heavily on α-renaming and, hence, are computationally quite involved [PV05].

The last option is to statically enforce a well-formedness condition that ensures static scope at run-time. This is the option we shall pursue in the following and, as we shall see, this choice simplifies the notions of substitution.

As always, the type of well-formed programs must be preserved by the semantics and a complication arises in this context:

The decision procedure, $\mathbf{C} \vdash P$, associated with well-formedness turns out to require knowledge about the free names of arbitrary sub-processes. This knowledge can only be specified in a satisfactory manner if environments are used to associate the free process identifiers with appropriate information. Thus, we shall specify well-formedness in terms of notions that are parametrised on environments as shown in Fig. 5.1.

The remainder of the chapter presents the required developments in more detail. In Section 5.1 we define the notions of free names and identifiers. Then, in Section 5.2, we specify the class of well-formed programs that is going to be the focus of later developments. In 5.3 we define the notions of name and identifier substitution for well-formed programs. In Section 5.4 we show a subject reduction result expressing that well-formedness is preserved by reaction. Finally, in Section 5.5, we conclude.

5.1 Free Names and Identifiers

In the following we shall formalise the free names of a process. In doing so we shall rely on the auxiliary definition of free and bound names of capabilities, which is shown in Table 5.1.

M	enter x	exit x	merge− x	$x!\{y\}$	$x\#!\{y\}$	$x\_!\{y\}$	$x\hat{\ }!\{y\}$
$\mathsf{bn}(M)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\mathsf{fn}(M)$	$\{x\}$	$\{x\}$	$\{x\}$	$\{x,y\}$	$\{x,y\}$	$\{x,y\}$	$\{x,y\}$
M	accept x	expel x	merge+ x	$x?\{p\}$	$x\#?\{p\}$	$x\hat{\ }?\{p\}$	$x\_?\{p\}$
$\mathsf{bn}(M)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{p\}$	$\{p\}$	$\{p\}$	$\{p\}$
$\mathsf{fn}(M)$	$\{x\}$	$\{x\}$	$\{x\}$	$\{x\}$	$\{x\}$	$\{x\}$	$\{x\}$

Table 5.1: Bound names, $\mathsf{bn}(M)$, and free names, $\mathsf{fn}(M)$, of capabilities M.

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

Table 5.2: Free names, $\mathsf{fn}_{\Gamma_{\mathsf{fn}}}(P)$, of processes, P.

Free names. As customary we specify the set of *free names*, $\mathsf{fn}_{\Gamma_{\mathsf{fn}}}(P) \in \mathcal{P}(\mathcal{N}ame)$, of a process, P, in terms of a recursive function that is defined as shown in Table 5.2.

The function is parametrised on an environment, $\Gamma_{\mathsf{fn}} : \mathbf{Pid} \to \mathcal{P}(\mathcal{N}ame)$, because later, when defining higher level notions such as well-formedness, we shall sometimes use it to specify the free names of sub-expressions where process identifiers occur free, e.g., a sub-process of the P in $\mathsf{rec}\,\mathrm{X}.\,P$. In this case the higher level notion can only be correctly defined if Γ_{fn} is used to associate each free process identifier, such as X, with the set of names occurring free in the corresponding process body, P.

The definition is pretty straightforward. The interesting cases are those of *restriction*, $(n)\,P$, where the bound name n is subtracted from the set of free names synthesised from the sub-process P, and *summation*, $\sum_{i\in I} M_i^{\ell_i}\,.\,P_i$, where the free names are collected from all capabilities M_i and bound names are subtracted, as appropriate, from the sets synthesised from the subterms P_i.

In the case of recursive processes, $\mathsf{rec}\,\mathrm{X}.\,P$, the corresponding bound process identifier, X, is wiped from the environment. The associated information is no

$$
\begin{aligned}
\mathsf{fpi}((n)\,P) &= \mathsf{fpi}(P)\\
\mathsf{fpi}([\,P\,]^{\mu}) &= \mathsf{fpi}(P)\\
\mathsf{fpi}(P \mid Q) &= \mathsf{fpi}(P) \cup \mathsf{fpi}(Q)\\
\mathsf{fpi}(\sum_{i\in I} M_i^{\ell_i}\,.\,P_i) &= \bigcup_{i\in I} \mathsf{fpi}(P_i)\\
\mathsf{fpi}(\mathsf{rec}\,\mathrm{X}.\,P) &= \mathsf{fpi}(P) \setminus \{X\}\\
\mathsf{fpi}(X) &= \{X\}
\end{aligned}
$$

Table 5.3: The free process identifiers, $\mathsf{fpi}(P)$, of processes, P.

longer needed because the present invocation of $\mathsf{fn}_{\Gamma_{\mathsf{fn}}}$ has access to the full body, P, of the recursive process. The latter suffices because a) we are computing a set rather than a multiset, and b) every syntactic entity of interest must occur in P in order to occur in $\mathsf{rec}\,\mathrm{X}.\,P$.

Clearly, if the process P of interest is identifier closed then the contents of Γ_{fn} is irrelevant, i.e., $\mathsf{fn}_{\Gamma_{\mathsf{fn}}}(P) = \mathsf{fn}_{[\,]}(P)$ for any Γ_{fn}. This lead to the following convention:

Convention 5.1 When P is known to be an identifier closed process it suffices to write $\mathsf{fn}(P)$ to denote the free names of P.

Free process identifiers. We now turn to the set of *free process identifiers*, $\mathsf{fpi}(P)$, of P, formally defined in Table 5.3. In this case no environment is required because recursive processes are 'self-closed', i.e., the substitution of a process identifier always results in an identifier closed process.

The only interesting cases are those of the recursive process, $\mathsf{rec}\,\mathrm{X}.\,P$, and the process identifier, X. In the former case X is subtracted from the set of free process identifiers synthesised from the sub-process P because it is bound. In the later case X is added to the set of free process identifiers.

5.2 Well-formed and Initial Programs.

The BioAmbients language is very expressive, but we do not consider all processes equally acceptable. Some are unacceptable because they may cause name or identifier capture; these are semantically ill-behaved. Some are unacceptable

$$\textsc{wf-res}\quad \frac{\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P}{\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} (n)\, P} \text{ if } \lfloor n \rfloor \in \mathbf{C}$$

$$\textsc{wf-amb}\quad \frac{\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P}{\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} [\, P \,]^{\mu}} \text{ if } \mathsf{fpi}(P) = \emptyset$$

$$\textsc{wf-par}\quad \frac{\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P \qquad \mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P}{\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P \,|\, Q}$$

$$\textsc{wf-sum}\quad \frac{\forall i \in I : \mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P_i}{\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} \sum_{i \in I} M_i^{\ell_i} \,.\, P_i} \text{ if } \begin{array}{l} \forall i \in I : \\ (\lfloor \mathsf{bn}(M_i) \rfloor \cap \mathbf{C}) = \emptyset \end{array}$$

$$\textsc{wf-rec}\quad \frac{\mathbf{C} \vdash_{\Gamma'_{\mathsf{fn}}} P}{\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} \mathsf{rec}\,\mathrm{X}.\, P} \text{ if } \; X \in \mathsf{fpi}(P) \wedge \Gamma'_{\mathsf{fn}} = \Gamma_{\mathsf{fn}}[X \mapsto \mathsf{fn}_{\Gamma_{\mathsf{fn}}[X \mapsto \emptyset]}(P)] \wedge \mathsf{nocap}_{\Gamma_{\mathsf{fn}}}$$

$$\textsc{wf-pid}\quad \mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} X$$

where we write $\mathsf{nocap}_{\Gamma_{\mathsf{fn}}}$ for

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

Table 5.4: Well-formedness, $\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P$, of a process, P, with respect to a set of constants, $\mathbf{C}$.

because they do not follow the conventions that ensure tractability of process name spaces; these are hard to analyse statically.

Well-formed Processes. We are now going to formally define the set of BioAmbients processes that we find acceptable. We shall say that a process P is *well-formed with respect to* $\mathbf{C}$ if it satisfies the well-formedness predicate $\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P$, defined by the inference system of Table 5.4. The notion of well-formedness captured by this definition encapsulates a number of important choices:

We require that processes observe the implicit typing requirements imposed by the distinction between constants (in $\mathbf{C}$) and variables (in $\mathbf{V}$) in **Name**. 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 $\mathbf{C}$.

We disallow infinite nesting of ambients. Technically, this kind of behaviour is hard to handle in a static analysis [NNP04] and, since it has no biological relevance either, we omit it from well-formed processes. The rule WF-AMB ensures this by demanding that the P in $[\ P\ ]^{\mu}$ is process identifier closed. A similar choice was made for Mobile Safe Ambients in [LS03].

We disallow useless recursive definitions. When the body P of a recursive definition $\mathsf{rec}\,\mathrm{X}.\,P$ does not make direct use of the associated process identifier X, i.e., $X \notin \mathsf{fpi}(P)$, then the recursive definition is useless. The rule WF-REC handles this by demanding that $X \in \mathsf{fpi}(P)$.

Finally, we intend processes to have static scope, and hence we have to ensure that identifier and name capture cannot occur. 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, $\mathsf{rec}\,\mathrm{X}.\,P$, can only recur through a name binder, e.g., $P = \cdots .\, n?\{p\}^{\ell} .\cdots .\, X$, if P has no free occurrence of the bound name, i.e., p. This is ruled out by the nocap part of rule WF-REC and prevents situations such as

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

and

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

Convention 5.2 When P is known to be identifier closed it suffices to write $\mathbf{C} \vdash P$, rather than $\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P$, for the well-formedness of P with respect to $\mathbf{C}$. ■

Programs. The well-formedness predicate gives us the means to distinguish well-behaved processes. This allows us to formally define the notion of *programs* – the processes $P_\star$ that satisfy the predicate $\mathsf{PRG}_{\mathbf{C}}(P_\star)$ defined as the conjunction of the following conditions:

- $P_\star$ is process identifier closed: $\mathsf{fpi}(P_\star) = \emptyset$.
- $P_\star$ has free names only from the constants: $\lfloor \mathsf{fn}(P_\star) \rfloor \subseteq \mathbf{C}$.
- $P_\star$ is well-formed with respect to the constants: $\mathbf{C} \vdash P_\star$.

Convention 5.3 In the following we shall assume that all process expressions considered are part of a (well-formed) program. ■

Remark 5.4 (Initial Programs) As a matter of convenience we shall also assume that programs, $P_\star$, subjected to static analysis, are *initial* in the sense that they satisfy $P_\star = \lfloor P_\star \rfloor$, where $\lfloor P \rfloor$ is the process that is as P except that every name x is replaced by the corresponding canonical name $\lfloor x \rfloor$.

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

Table 5.5: Substitution, $P[{}^{Q}/_{X}]$, of a process Q for an identifier X in a process P.

5.3 Substitution of Identifiers and Names.

We now turn to the formal definitions of *substitution.*

Substitution of Process Identifiers. We shall first define what it means to substitute a process expression Q for a process identifier X in a process expression P. As pointed out in Section 5.2 the risk of name or identifier capture arises in conjunction with identifier substitution. As evident from the reaction semantics of Section 3.2, however, a substitution, $P[{}^{Q}/_{X}]$, always arises from the unfolding of the top-most recursive process; hence it is safe to assume that

- Q is the top-level recursive process that defines X, i.e., $Q = \mathsf{rec}\,\mathrm{X}.\,Q'$.
- 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 Table 5.5.

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

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

Table 5.6: Substitution, $P[{}^{m}/_{x}]$, of a constant m for a name x in a process P.

of the separation of **Name** into **C** and **V**.

Similarly the substitution over summations is correctly defined because $\mathsf{bn}(M_i) \cap \mathsf{fn}_{\Gamma_{\mathsf{fn}}}(Q) \neq \emptyset$ cannot be the case if $X \in \mathsf{fpi}(P)$.

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.

Substitution of Names. We shall now go on to define what it means to substitute a constant m for an arbitrary name x in a program P. As pointed out in Section 3.1.2, name substitution can lead to name capture. In the context convention 5.3, however, this is relevant only to constants; this facilitates the definition of Table 5.6, which does not rely on α-conversion of variables.

The substitution over restrictions is defined in the standard way. In case of constant capture we apply disciplined α-conversion in order to avoid the capture.

The substitution over summations is simpler. Capture cannot occur in the context of well-formed programs due to the separation of **Name** into **C** and **V**. Of course, we define the associated notion of substitution over capability prefixes $M_i^{\ell_i}[{}^{m}/_{x}]$ such that *bound* variables $p \in \mathsf{bn}(M_i)$ 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 substitution over process identifiers straightforward.

5.4 Properties of Programs

Later, when proving properties of the analyses, we shall rely on the well-formedness conditions of programs. We can do this because programs evaluate into programs, i.e.,

$$\text{If } \mathsf{PRG}_{\mathbf{C}}(P_\star) \text{ and } P_\star \xrightarrow{\tilde{L}}{}^\star P \text{ then } \mathsf{PRG}_{\mathbf{C}}(P)$$

In order to show this we shall first show a number of minor results.

Free Names. We start by showing some basic properties of free names:

Fact 5.5 Assume $Q = \mathsf{rec}\,\mathrm{X}.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

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

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 $\mathsf{fpi}(Q) = \emptyset \Rightarrow (\mathsf{fn}_{\Gamma_{\mathsf{fn}}}(Q) = \mathsf{fn}(Q))$ for all Γ_{fn}. □

Fact 5.6 $\mathsf{fn}_{\Gamma_{\mathsf{fn}}}(P[{}^{m}/{}_{x}]) = \begin{cases}(\mathsf{fn}_{\Gamma_{\mathsf{fn}}}(P) \setminus \{x\}) \cup \{m\} & \text{if } x \in \mathsf{fn}_{\Gamma_{\mathsf{fn}}}(P)\\ \mathsf{fn}_{\Gamma_{\mathsf{fn}}}(P) & \text{otherwise}\end{cases}$

Proof By structural induction on P. □

Fact 5.7 If $\mathsf{fpi}(P) = \emptyset$ and $\mathbf{C} \vdash P$ then both of the following hold:

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

2. 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 5.6. In the case of H-UREC we use Fact 5.5.

The proof of *(2)* follows by induction on the inference of $P \xrightarrow{\tilde{\ell}} Q$, where we use Fact 5.6 for the communication axioms and *(1)* in conjunction with the induction hypothesis in the case of R-AUX. □

Free Process Identifiers. We establish a similar result for the free process identifiers:

Fact 5.8 If $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

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

2. 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. □

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

Fact 5.9 Assume $Q = \mathsf{rec}\, X.\, Q'$ and $\mathsf{fpi}(Q) = \emptyset$; if furthermore $P \prec Q$ then

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

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

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 5.11-*(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

$$\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} (\mathsf{rec}\,\mathrm{Y}.\,P)[{}^{Q}/_{X}]$$

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

$$\mathbf{C} \vdash_{\Gamma'_{\mathsf{fn}}} \mathsf{rec}\,\mathrm{Y}.\,P \text{ and} \tag{1}$$
$$\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} Q \tag{2}$$

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

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

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

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

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

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

Now we observe that

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

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

From (III) we get

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

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

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

However, it follows from simple calculations and Fact 5.5 that $\Gamma_{\mathsf{fn}}''' = \Gamma_{\mathsf{fn}}'''' = \mathsf{fn}_{\Gamma_{\mathsf{fn}}'}(\mathsf{rec}\, Y.\, P)$; 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 proof of $\Leftarrow$ follows by similar considerations. □

Fact 5.10 If $\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P$ and $\lfloor m \rfloor \in \mathbf{C}$ then $\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P[{}^{m}/_{x}]$

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

1. If $P \Rrightarrow Q$ then $\mathbf{C} \vdash Q$
2. 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 5.9. In case of H-CAMB we use Fact 5.8-*(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 5.10. In the case of R-AMB we use Fact 5.8-*(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 5.12 (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 Fact 5.11, Fact 5.7, and Fact 5.8. □

Corollary 5.13 (Type soundness) Assume that $P_\star$ is a well-formed initial program then:

$$\text{If } P_\star \xrightarrow{\tilde{L}}{}^{\star} P \xrightarrow{\tilde{\ell}} Q \text{ then } \mathsf{PRG}_{\mathbf{C}}(P) \text{ and } \mathsf{PRG}_{\mathbf{C}}(Q).$$

Proof The result follows by induction of the length of the derivation sequence $\tilde{L}$, where we use Fact 5.12 to establish the base case and, in conjunction with the induction hypothesis, to establish the inductive step. □

5.5 Concluding Remarks

This chapter has introduced a notion of well-formed programs that can be statically checked and is invariant under reaction. This notion of well-formedness helps us achieve simple definitions of substitution that rely on neither heavy use of α-renaming [PV05] nor parametrised process constants [SW01, Cai04]. Furthermore, well-formed programs do not exhibit behaviours, such as unbounded nesting, that we find incompatible with the biological domain.

In the following chapters we shall (mostly) restrict our attention to the class of well-formed processes. This helps us formulate and show the required correctness results; hence this is the class of programs for which we *guarantee* the analyses to be correct. The running example and the two case models all qualify as well-formed programs.

Chapter 6

Context Insensitive Control Flow Analysis

"Seek simplicity but distrust it."

— Alfred North Whitehead

This chapter presents a *context insensitive* or *mono-variant Control Flow Analysis* (*0CFA*) for the BioAmbients language, and evaluates it by analysing the LDL degradation pathway of Section 3.3. Much of the presented material was previously covered in [NNPdR07].

The Control Flow Analysis is defined as a Flow Logic

$$0\text{CFA} = \mathcal{FL}(\text{BioAmbients}, \textsc{alfp})$$

that aims to safely over-approximate the set of spatial configurations that may arise at run-time. Thus, the focus of the approximation is on the spatial hierarchy established by the nesting of ambient boundaries. More precisely, the result of application of the analysis to a program $P_\star$ yields an over-approximation of

1. the set of roles and capabilities that might show up in each role and
2. the set of constants that might be bound to each variable.

The approach of the analysis bears some resemblance to abstract interpretation: Analysis results are acceptable if they, within the limited precision of the analysis domain,

1. faithfully represent the initial configuration and

$$\begin{array}{rcl}
\mathsf{cap}_{\Gamma_{\mathsf{cap}}}((n)\,P) & = & \mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P) \\
\mathsf{cap}_{\Gamma_{\mathsf{cap}}}([\![\, P\,]\!]^{\mu}) & = & \mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P) \\
\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P \,\big|\, Q) & = & \mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P) \cup \mathsf{cap}_{\Gamma_{\mathsf{cap}}}(Q) \\
\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(\sum_{i\in I} M_i^{\ell_i} \,.\, P_i) & = & \bigcup_{i\in I}(\{\ell_i\} \cup \mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P_i)) \\
\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(\mathsf{rec}\,\mathrm{X}.\,P) & = & \mathsf{cap}_{\Gamma_{\mathsf{cap}}[X\mapsto\emptyset]}(P) \\
\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(X) & = & \Gamma_{\mathsf{cap}}(X)
\end{array}$$

Table 6.1: Occurring capabilities, $\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P)$, of a process P.

2. are closures with respect to a set of conditions that mimic the semantics.

In most cases the closure conditions contribute the bulk of the analysis information and, hence, it is well worthwhile to be as precise as possible and limit the scope of the closure conditions to the set of capabilities that actually have a chance of becoming *concurrently possible* at run-time. This is captured by a statically computed auxiliary relation $\mathsf{CP}_\star$.

The chapter is in four sections. First, in Section 6.1, we define the auxiliary relation $\mathsf{CP}_\star$, and show that it constitutes a safe approximation to the set of concurrently possible capabilities. Then, in Section 6.2, we specify the 0CFA analysis as a Flow Logic, show that it is correct, and how to compute it. In Section 6.3 we apply the 0CFA to the LDL degradation pathway in order to both evaluate the analysis and obtain information about the pathway. Finally, in Section 6.4 we summarise our findings.

6.1 Concurrently Possible Capabilities

For any given program, $P_\star$, the problem of deciding the precise set of capability interactions that may come to pass is undecidable. From a semantic point of view, however, two capabilities can react only if simultaneously present in the same subordinate solution [BDPZ03]. In the following we shall use this insight to compute a first crude approximation to the set of potential capability interactions. This approximation will be useful later when we specify the 0CFA.

We start by defining the set of *occurring capabilities*, $\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P) \in \mathcal{P}(\mathbf{Lab})$, of a process P as shown in Table 6.1. Like the free names of the previous chapter, the set is parameterised on an environment, $\Gamma_{\mathsf{cap}} : \mathbf{Pid} \to \mathcal{P}(\mathbf{Lab})$, intended to associate each process identifier occurring free in P with the set of capabilities occurring in the corresponding process body. The definition is

$$
\begin{array}{rcl}
\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}((n)\,P) & = & \mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(P) \\
\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}([\,P\,]^{\mu}) & = & \mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(P) \\
\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(P \mid Q) & = & \mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(P) \cup \mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(Q) \cup \\
 & & (\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P) \times \mathsf{cap}_{\Gamma_{\mathsf{cap}}}(Q)) \cup \\
 & & (\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(Q) \times \mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P)) \\
\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(\sum_{i\in I} M_i^{\ell_i}\,.\,P_i) & = & \bigcup_{i\in I}(\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(P_i)) \\
\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(\mathsf{rec}\,\mathrm{X}.\,P) & = & \mathsf{CP}_{\Gamma_{\mathsf{cap}}[X\mapsto \mathsf{cap}_{\Gamma_{\mathsf{cap}}[X\mapsto\emptyset]}(P)],\Delta_{\mathsf{CP}}[X\mapsto\emptyset]}(P) \\
\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(X) & = & \Delta_{\mathsf{CP}}(X)
\end{array}
$$

Table 6.2: Concurrently possible capabilities, $\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(P)$, of a process P.

straightforward — the associated function simply performs a recursive traversal of process expressions. Each occurring prefix label is picked up as defined in the case for guarded sums.

Intuitively, two capabilities can interact only if separated by a '$\mid$'. Furthermore, they cannot possibly interact if separated by a '+' [BDPZ03]. We use these insights to specify the set of *concurrently possible capabilities*, $\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(P)$, of a BioAmbients process, P, as shown in Table 6.2. The set is parameterised on two environments:

The environment Γ_{cap} helps in making the distinction between '$\mid$' and '+'. In the case of a parallel composition, $P \mid Q$, the auxiliary function $\mathsf{cap}_{\Gamma_{\mathsf{cap}}}()$ is used to record that prefixes in the two branches may interact with one another. In contrast, no such information is recorded for non-deterministic choice, $P + Q$, i.e., summation.

The environment $\Delta_{\mathsf{CP}} : \mathbf{Pid} \rightarrow \mathcal{P}(\mathbf{Lab} \times \mathbf{Lab})$ simply helps us define $\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}()$ in such a way that it can be applied to sub-expressions that are not identifier closed.

Thus, in the case of a recursive process, $\mathsf{rec}\,\mathrm{X}.\,P$, the environment Γ_{cap} is updated to associate X with the capabilities occurring in P, whereas the environment Δ_{CP} is updated to ensure that X is associated with no information.

Convention 6.1 When P is known to be identifier closed we write $\mathsf{CP}(P)$, rather than $\mathsf{CP}_{[\,],[\,]}(P)$, to denote the concurrently possible capabilities of P. When furthermore the subject of approximation is an initial program, $P_\star$, we write $\mathsf{CP}_\star$ for $\mathsf{CP}(P_\star)$. ■

Example 6.2 For the running example P_{eat} we obtain the relation $\mathsf{CP}_{\mathsf{eat}}$ shown below:

$$\begin{array}{l}
(\ell_1,\ell_2),(\ell_1,\ell_3),(\ell_1,\ell_4),(\ell_1,\ell_5),(\ell_1,\ell_6),(\ell_1,\ell_7),(\ell_1\ell_8),(\ell_1,\ell_9),\\
(\ell_2,\ell_1),(\ell_2,\ell_6),(\ell_2,\ell_7),(\ell_2,\ell_8),(\ell_2,\ell_9),\\
(\ell_3,\ell_1),(\ell_3,\ell_6),(\ell_3,\ell_7),(\ell_3,\ell_8),(\ell_3,\ell_9),\\
(\ell_4,\ell_1),(\ell_4,\ell_6),(\ell_4,\ell_7),(\ell_4,\ell_8),(\ell_4,\ell_9),\\
(\ell_5,\ell_1),(\ell_5,\ell_6),(\ell_5,\ell_7),(\ell_5,\ell_8),(\ell_5,\ell_9),\\
(\ell_6,\ell_1),(\ell_6,\ell_2),(\ell_6,\ell_3),(\ell_6,\ell_4),(\ell_6,\ell_5),(\ell_6,\ell_7),(\ell_6\ell_8),(\ell_6,\ell_9),\\
(\ell_7,\ell_1),(\ell_7,\ell_2),(\ell_7,\ell_3),(\ell_7,\ell_4),(\ell_7,\ell_5),(\ell_7,\ell_6),\\
(\ell_8,\ell_1),(\ell_8,\ell_2),(\ell_8,\ell_3),(\ell_8,\ell_4),(\ell_8,\ell_5),(\ell_8,\ell_6),\\
(\ell_9,\ell_1),(\ell_9,\ell_2),(\ell_9,\ell_3),(\ell_9,\ell_4),(\ell_9,\ell_5),(\ell_9,\ell_6)
\end{array}$$

As must be expected, this constitutes a crude over-approximation of the interactions that may take place. □

Soundness of $\mathsf{CP}_\star$. We shall now show that for any $P_\star$ the corresponding $\mathsf{CP}_\star$ constitutes a safe over-approximation, i.e.,

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

The result rests on a number of minor results:

Fact 6.3 Assume $Q = \mathsf{rec}\,\mathrm{X}.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

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

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

Facts 6.4 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

1. If $P \Rrightarrow Q$ then $\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P) \supseteq \mathsf{cap}_{\Gamma_{\mathsf{cap}}}(Q)$
2. If $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P) \supseteq \mathsf{cap}_{\Gamma_{\mathsf{cap}}}(Q)$

Proof For (1) the proof proceeds by induction on the shape of the proof tree establishing $P \Rrightarrow Q$, using Fact 6.3 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 6.5 Assume $Q = \mathsf{rec}\, X.\, Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

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

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

Proof The proof proceeds by structural induction on P. In the cases of parallel compositions and recursive processes it uses Fact 6.3. □

Facts 6.6 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

1. If $P \Rrightarrow Q$ then $(\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(P) \supseteq \mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(Q))$
2. If $P \xrightarrow{\tilde{\ell}} Q$ then $(\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(P) \supseteq \mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}(Q))$

Proof For (1) the proof proceeds by induction on the shape of the proof tree establishing $P \Rrightarrow Q$, using Fact 6.5 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 6.7 (Subject reduction)
If $\mathsf{PRG}_{\mathbf{C}}(P)$ and $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{CP}(P) \supseteq \mathsf{CP}(Q)$. □

Finally this thread of reasoning leads to the sought result:

Lemma 6.8 ($\mathsf{CP}_\star$ 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 6.7, and the inductive step follows from the induction hypothesis in conjunction with Corollaries 5.13 and 6.7. □

6.2 Control Flow Analysis

We now turn to the task of approximating the set of spatial configurations that may arise at run-time. For this purpose we shall specify a context independent Control Flow Analysis (0CFA).

6.2.1 The Analysis Domain

For a given BioAmbients program $P_\star$ we want the analysis to specify the following three components:

- An approximation of the relevant name bindings:
$$\mathcal{R} \subseteq \mathbf{V} \times \mathbf{C}$$
where we write $n \in \mathcal{R}(p)$ or $(p, n) \in \mathcal{R}$ to assert the truth of predicate $\mathcal{R}(p, n)$, i.e., that $\mathcal{R}$ records that the variable p might become bound to the constant n.

- An approximation of the contents (ambients, prefixes) of ambients:
$$\mathcal{I} \subseteq \mathbf{Role} \times (\mathbf{Role} \cup (\mathbf{Cap} \times \mathbf{Lab}))$$
where we write $\mu \in \mathcal{I}(\mu')$ or $(\mu', \mu) \in \mathcal{I}$ (or, $M^\ell \in \mathcal{I}(\mu')$ or $(\mu', M^\ell) \in \mathcal{I}$) to denote the truth of predicate $\mathcal{I}(\mu', \mu)$ (or $\mathcal{I}(\mu', M^\ell)$), i.e., that $\mathcal{I}$ records that an ambient of role μ (or a prefix M^ℓ) might occur inside an ambient of role μ'.

- An approximation of the pairs of capabilities that may react:
$$\mathcal{F} \subseteq \mathbf{Lab} \times \mathbf{Lab}$$
where we shall write $(\ell_1, \ell_2) \in \mathcal{F}$ to denote the truth of $\mathcal{F}(\ell_1, \ell_2)$, i.e., that $\mathcal{F}$ records that prefixes labelled ℓ_1 and ℓ_2 might react.

The *domain* of the analysis is the direct product of those corresponding to the three components. This clearly constitutes a complete lattice under the component-wise subset ordering.

Remark 6.9 Note that in the context of a concrete program $P_\star$ the domain specialises to a finite lattice over $\mathbf{V}_\star, \mathbf{C}_\star, \mathbf{Role}_\star, \mathbf{Cap}_\star$, and $\mathbf{Lab}_\star$, which are all finite sets. Hence all instances satisfy the ascending chain condition. ■

$(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} (n)\, P$	iff	$(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P$
$(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} [\, P\,]^{\mu_c}$	iff	$\mu_c \in \mathcal{I}(\mu) \wedge (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu_c} P$
$(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P \mid Q$	iff	$(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P \wedge (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} Q$
$(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} \sum_{i\in I} M_i^{\ell_i} . P_i$	iff	$\forall i \in I : (\lfloor M_i \rfloor^{\ell_i} \in \mathcal{I}(\mu) \wedge$ $\mathsf{closure}_{\lceil M_i \rceil} \wedge (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P_i)$
$(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} \mathsf{rec}\, \mathrm{X}.\, P$	iff	$(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P$
$(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} X$	iff	true

Table 6.3: The 0CFA acceptability judgement.

6.2.2 The Acceptability Judgement

The *acceptability judgement* of the analysis takes the form

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

and expresses that, when the sub-process P (of $P_\star$) is enclosed within an ambient of role $\mu \in$ **Role**, then $\mathcal{I}$, $\mathcal{R}$, and $\mathcal{F}$ correctly capture the behaviour of P — meaning that $\mathcal{I}$ approximates the contents that may occur in each ambient, $\mathcal{R}$ the bindings of names that may take place, and $\mathcal{F}$ the capability pairs that may react, as P evolves inside $P_\star$.

The judgement is specified in Table 6.3 and refers to Table 6.4 and Table 6.5 for specifications of the closure conditions, $\mathsf{closure}_{\lceil M \rceil}$, where $\lceil M \rceil$ is as M but with names replaced by '$\cdot$'. The specification is syntax directed, which makes the 0CFA *compositional* in the terminology of Flow Logic.

The clauses of the definition carry the following meaning:

Name restriction: An analysis estimate $(\mathcal{I},\mathcal{R},\mathcal{F})$ is acceptable for a process, $(n)\, P$, located inside an ambient of role μ if, and only if, acceptable for the sub-process P in μ. Avoiding further requirements helps us ensure that the analysis is invariant under, rather than preserved by, the heating relation, even when H-EVA is destabilising the set of name restrictions.

Ambient boundary: An analysis estimate $(\mathcal{I},\mathcal{R},\mathcal{F})$ is acceptable for a process, $[\, P\,]^{\mu_c}$, located inside an ambient of role μ if, and only if,

- $(\mu, \mu_c) \in \mathcal{I}$ and
- $(\mathcal{I},\mathcal{R},\mathcal{F})$ is acceptable for the sub-process P in μ_c.

Parallel composition: An estimate $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ is acceptable for $P \,\big|\, Q$ in μ if, and only if, acceptable for both P and Q 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 by using $\mathsf{CP}_\star$.

Summation: An estimate $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ 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, $\mathsf{closure}_{\lceil M_i \rceil}$, is satisfied by $(\mathcal{I}, \mathcal{R}, \mathcal{F})$, and
- $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ is acceptable for the associated sub-process P_i in μ.

Recursive process: An analysis estimate $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ is acceptable for $\mathsf{rec}\, \mathrm{X}. P$ in μ if, and only if, acceptable for the sub-process P in μ.

Process identifier: Any $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ is an acceptable analysis estimate for X in μ, hence we ignore them.

This simple treatment of recursion is acceptable because:

- The well-formedness conditions ensure 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.
- Furthermore, the flow-insensitivity of the CFA, i.e., that prefix sequences are represented as sets rather than lists, ensures that a single inspection of the body suffices.

6.2.3 The Closure Conditions

Acceptable 0CFA estimates take the dynamic behaviour of capabilities into account. This is ensured by a set of closure conditions that mimic, within the limited precision of the analysis domain, the axioms of the reaction relation. Acceptable estimates are closures with respect to these conditions: whenever an estimate satisfies the pre-conditions it must also satisfy the conclusion.

In dealing with these requirements the conditions rely on the $\mathcal{R}$ component, which records only the potential bindings of variables to names and does not include any information about constants. In order to provide the required information about constant definitions we introduce a new relation

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

Table 6.4: 0CFA closure conditions for movement.

$$\langle\mathcal{R}\rangle \subseteq \mathbf{Name} \times \mathbf{C}$$

defined by the following closure condition:

$$\mathsf{complete}\mathcal{R} = (\forall p, m : \mathcal{R}(p, m) \Rightarrow \langle\mathcal{R}\rangle(p, m)) \wedge (\forall n : n \in \mathbf{C} \Rightarrow \langle\mathcal{R}\rangle(n, n))$$

This condition asserts that $\langle\mathcal{R}\rangle$ contains everything that $\mathcal{R}$ does, and that $\langle\mathcal{R}\rangle$ binds all constants to themselves. Thus $\langle\mathcal{R}\rangle$ acts exactly like $\mathcal{R}$ with respect to variable bindings but also acts as the identity on constants.

Movement Closures. The closure conditions for movement ensure that analysis estimates are only acceptable if they reflect the potential consequences of movements that might take place at run-time. If, e.g., $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ implies that an enter movement might become enabled in some ambient (role) μ (Fig. 6.1), i.e.,

- An enter and an accept capability might occur in sibling ambients:

$$\begin{array}{l} \mathsf{enter}\ x^{\ell_1} \in \mathcal{I}(\mu_1) \wedge \mu_1 \in \mathcal{I}(\mu) \wedge \\ \mathsf{accept}\ y^{\ell_2} \in \mathcal{I}(\mu_2) \wedge \mu_2 \in \mathcal{I}(\mu) \end{array}$$

- The corresponding prefixes might be concurrently possible:

$$(\ell_1, \ell_2) \in \mathsf{CP}_\star$$

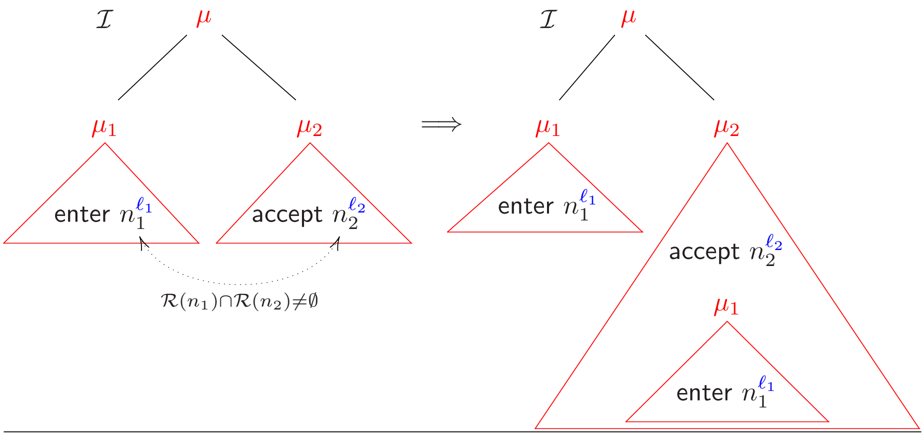


Figure 6.1: 0CFA closure condition for enter movement.

- The enter and accept capabilities might agree on a communication channel:

$$\langle\mathcal{R}\rangle(x) \cap \langle\mathcal{R}\rangle(y) \neq \emptyset$$

Remark 6.10 Note that the capabilities may reference the channel name directly, by using a constant, or indirectly, by using a variable. Both situations are captured when the condition is expressed as a requirement of non-empty intersection in $\langle\mathcal{R}\rangle$ rather than $\mathcal{R}$. ■

Then $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ should reflect that:

- The moving ambient role μ_1 might occur in an ambient of role μ_2:

$$\mu_1 \in \mathcal{I}(\mu_2)$$

- The corresponding capabilities, ℓ_1 and ℓ_2, might react:

$$(\ell_1, \ell_2) \in \mathcal{F}$$

Communication Closures. The closure conditions for communication ensure that analysis estimates are only acceptable if they reflect the potential consequences of communications that might take place at run-time. If, for example, $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ implies that a local communication might become enabled in some ambient (role) μ (Fig. 6.2):

$$
\begin{array}{lcl}
\mathsf{closure}_{\cdot!\{\cdot\}} & = & \forall \mu, x, y, z, p, \ell_1, \ell_2 : \\
& & \quad x!\{z\}^{\ell_1} \in \mathcal{I}(\mu) \wedge \\
& & \quad 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?\{\cdot\}} & = & \mathsf{true} \\
\mathsf{closure}_{\cdot\_!\{\cdot\}} & = & \forall \mu, \mu_1, x, y, z, p, \ell_1, \ell_2 : \\
& & \quad x\_!\{z\}^{\ell_1} \in \mathcal{I}(\mu) \wedge \\
& & \quad 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}_{\cdot\hat{}?\{\cdot\}} & = & \mathsf{true} \\
\mathsf{closure}_{\cdot\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 \\
& & \quad 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\_?\{\cdot\}} & = & \mathsf{true} \\
\mathsf{closure}_{\cdot\#!\{\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 \\
& & \quad 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\#?\{\cdot\}} & = & \mathsf{true}
\end{array}
$$

Table 6.5: 0CFA closure conditions for communication.

- A local input action and a local output action might occur in the same context:

$$x!\{z\}^{\ell_1} \in \mathcal{I}(\mu) \wedge y?\{p\}^{\ell_2} \in \mathcal{I}(\mu)$$

- The corresponding prefixes might be concurrently possible:

$$(\ell_1, \ell_2) \in \mathsf{CP}_\star$$

- The input and output actions might agree on a communication channel:

$$\langle\mathcal{R}\rangle(x) \cap \langle\mathcal{R}\rangle(y) \neq \emptyset$$

Then $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ should reflect that:

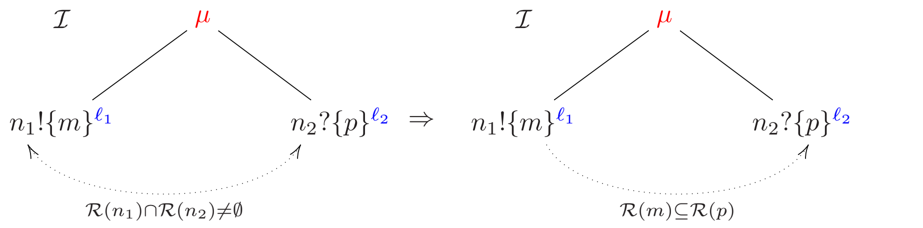


Figure 6.2: 0CFA closure condition for local communication.

- Any name that might become bound to z might become the object of the communication and might thus become bound to p:

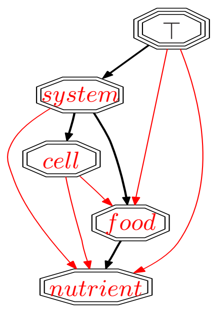
$$\langle\mathcal{R}\rangle(z) \subseteq \mathcal{R}(p)$$

- The corresponding capabilities, ℓ_1 and ℓ_2, might react:

$$(\ell_1, \ell_2) \in \mathcal{F}$$

Remark 6.11 Note that, while semantically z is always a constant, it may be either a constant or a variable in the representation of $(\mathcal{I}, \mathcal{R}, \mathcal{F})$. In contrast p always denotes a variable. This is why the inclusion requirement is expressed with $\langle\mathcal{R}\rangle$ as the source and $\mathcal{R}$ as the target. ■

Example 6.12 The best analysis estimate $(\mathcal{I}_{\mathsf{eat}}, \mathcal{R}_{\mathsf{eat}}, \mathcal{F}_{\mathsf{eat}})$ of the running example P_{eat} is shown below:



n	$\mathcal{R}(n)$
rl	RL

$\mathcal{R}_{\mathsf{eat}}$ component

ℓ	$\mathcal{F}(\ell)$
ℓ_7	ℓ_2
ℓ_4	ℓ_1, ℓ_8
ℓ_6	ℓ_3
ℓ_5	ℓ_9

$\mathcal{I}_{\mathsf{eat}}$ component

$\mathcal{F}_{\mathsf{eat}}$ component

In the figure the contents of $\mathcal{I}_{\mathsf{eat}}$ is presented graphically, while $\mathcal{R}_{\mathsf{eat}}$ and $\mathcal{F}_{\mathsf{eat}}$ 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, $\top$, used as superscript in $(\mathcal{I}_{\mathsf{eat}}, \mathcal{R}_{\mathsf{eat}}, \mathcal{F}_{\mathsf{eat}}) \models^{\top} P_{\mathsf{eat}}$, the double bordered nodes connected by bold (black) edges represent the initial configuration (as specified by Table 6.3), and the remaining (red) edges represent the system dynamics (as specified by Table 6.4, 6.5, and 6.2). The trees of the individual frames of Example 3.28 are all sub-trees of these graphs.

Considering the semantics of P_{eat} (Example 3.28) it should be clear that nutrients can only be released from the food particles in the context of the cell. The analysis result, however, exhibits poor precision and is not able to reach this conclusion. $\square$

6.2.4 Properties of the 0CFA

We have now *defined* an acceptability judgement that specifies the notion of an acceptable CFA estimate $(\mathcal{I}, \mathcal{R}, \mathcal{F})$. It remains, to be *shown* that this judgement does indeed specify a static analysis that is 1) *well-defined*, 2) *sound with respect to the semantics*, and 3) *implementable*. This we shall do in the following.

Well-definedess. Due to the compositional nature of the specification the first property is easy to establish:

Theorem 6.13 (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.

Proof The proof proceeds by straightforward structural induction on P. $\square$

This means that we can always check an a priori given analysis estimate.

Semantic Correctness. In order to establish the second property we have to show that the analysis is *correct* in the sense that the studied properties are"$\sqsupseteq$-preserved" under heating and reaction.

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

Fact 6.14 Assume $Q = \mathsf{rec}\, X.\, Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

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

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

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

Lemma 6.15 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then the following holds:

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

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 $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\mu} \lfloor P \rfloor \Leftrightarrow (\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\mu} \lfloor Q \rfloor$ follows by referential transparency. Finally, in the case of *unfolding of recursion* the desired result follows directly from Fact 6.14. $\square$

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

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

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

This expansion satisfies a useful substitution property.

Fact 6.16 If $\lfloor n \rfloor \in \mathcal{R}(x)$ and $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\mu} P$ then $(\mathcal{I}@\mathcal{R}, \mathcal{R}, \mathcal{F}) \models^{\mu} P[{}^{n}/_{x}]$.

Proof The proof proceeds by structural induction on P. $\square$

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

Lemma 6.17 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then, if furthermore $\mathsf{CP}(P) \subseteq \mathsf{CP}_{\star}$ the following holds:

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

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 $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\top} P$ and $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\top} Q$

using the definition of $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\mu} P$. In the case of communication we perform a similar expansion and then obtain the desired result using Fact 6.16. The remaining cases follow from the induction hypothesis and, in the case of R-AUX, uses Lemma 6.15. $\square$

Corollary 6.18 (Subject reduction) Assume $\mathsf{PRG}_{\mathbf{C}}(P)$ and $\mathsf{CP}(P) \subseteq \mathsf{CP}_{\star}$; if furthermore $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\mu} P$ and $P \xrightarrow{\tilde{\ell}} Q$ then $(\mathcal{I}@\mathcal{R}, \mathcal{R}, \mathcal{F}) \models^{\mu} Q$ and $\tilde{\ell} \in \mathcal{F}$. $\square$

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.

Analysis estimates are typically based on initial programs $P_{\star}$ related to Q through a longer reaction sequence of the form $P_{\star} \xrightarrow{\tilde{L}}{}^{\star} P \xrightarrow{\tilde{\ell}} Q$. It is immediate to show that $\mathcal{I}@\mathcal{R} = (\mathcal{I}@\mathcal{R})@\mathcal{R}$ and hence we can state the overall correctness result as follows:

Theorem 6.19 (Semantic correctness) Assume that $P_{\star}$ is a well-formed initial program, then $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\top} P_{\star}$ and $P_{\star} \xrightarrow{\tilde{L}}{}^{\star} P$ entails $(\mathcal{I}@\mathcal{R}, \mathcal{R}, \mathcal{F}) \models^{\top} P$ and $\tilde{L} \subseteq \mathcal{F}$.

Proof The corollary follows by induction on the length of $\tilde{L}$. The base case holds vacuously. The inductive step is established using Corollary 6.17, Corollary 5.13, the induction hypothesis, and the above insight. $\square$

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

Implementability. Having defined the notion of acceptable analysis estimates and established their semantical soundness we shall now show that there is always a best estimate and indicate how to compute it.

Recall from Section 4.3 that a Moore family results guarantees best estimates. Here we write best to mean *least* with respect to the partial order of the analysis domain:

$$(\mathcal{I}_1, \mathcal{R}_1, \mathcal{F}_1) \sqsubseteq (\mathcal{I}_2, \mathcal{R}_2, \mathcal{F}_2) \quad \text{iff} \quad (\mathcal{I}_1 \subseteq \mathcal{I}_2) \wedge (\mathcal{R}_1 \subseteq \mathcal{R}_2) \wedge (\mathcal{F}_1 \subseteq \mathcal{F}_2)$$

In the context of this ordering we can prove the theorem:

Theorem 6.20 (Moore family (0CFA)) For any program $P_{\star}$ the set of acceptable analyses under $\models^{\mu}$ constitutes a Moore family, i.e.,

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

Proof Assume that for some index set I and process P we have

$$\forall i \in I : (\mathcal{I}_i,\mathcal{R}_i,\mathcal{F}_i) \models^{\mu} P$$

It then remains to be shown that

$$\bigsqcap_{i\in I}(\mathcal{I}_i,\mathcal{R}_i,\mathcal{F}_i) \models^{\mu} P$$

This follows by structural induction on P.

Case $(n)\,P$**:** Follows by the induction hypothesis.

Case $[\,P\,]^{\mu}$**:** Assume that $\forall i \in I : (\mathcal{I}_i,\mathcal{R}_i,\mathcal{F}_i) \models^{\mu} [\,P\,]^{\mu_c}$. It follows that $\mu_c \in \mathcal{I}_i(\mu)$ and $(\mathcal{I}_i,\mathcal{R}_i,\mathcal{F}_i) \models^{\mu_c} P$ for all $i \in I$. The induction hypothesis ensures that $\bigsqcap_{i\in I}(\mathcal{I}_i,\mathcal{R}_i,\mathcal{F}_i) \models^{\mu} P$ and, since $\mu_c \in \mathcal{I}_i(\mu)$ for all $i \in I$, we also have $\mu_c \in \bigcap_{i\in I}\mathcal{I}_i(\mu)$. The desired result $\bigsqcap_{i\in I}(\mathcal{I}_i,\mathcal{R}_i,\mathcal{F}_i) \models^{\mu} [\,P\,]^{\mu}$ follows directly.

Case $P \mid Q$**:** follows from the induction hypothesis.

Case $\sum_{j\in J} M_j^{\ell_j}\,.\,P_j$**:** Assume that $\forall i \in I : (\mathcal{I}_i,\mathcal{R}_i,\mathcal{F}_i) \models^{\mu} \sum_{j\in J} M_j^{\ell_j}\,.\,P_j$. It follows that $\forall j \in J : (\lfloor M_j \rfloor^{\ell_j} \in \mathcal{I}_i(\mu) \wedge \mathsf{closure}_{\lceil M_j \rceil} \wedge (\mathcal{I}_i,\mathcal{R}_i,\mathcal{F}_i) \models^{\mu} P_j)$ for all $i \in I$. As before, $\bigsqcap_{i\in I}(\mathcal{I}_i,\mathcal{R}_i,\mathcal{F}_i) \models^{\mu} P_j$ for all $j \in J$ follows by the induction hypothesis, and $\lfloor M_j \rfloor^{\ell_j} \in (\bigcap_{i\in I}\mathcal{I}_i)(\mu)$ for all $j \in J$ follows because the formula is universally quantified.

It remains to be shown that $\mathsf{closure}_{\lceil M_j \rceil}$ for all $j \in J$ is satisfied by $\bigsqcap_{i\in I}(\mathcal{I}_i,\mathcal{R}_i,\mathcal{F}_i)$. This is done by case analysis on M, and we shall consider the case for the enter capability:
Fix $\mu, \mu_1, \mu_2, x, y, \ell_1, \ell_2$ such that

$$\begin{array}{l}\mathsf{enter}\ x^{\ell_1} \in (\bigcap_{i\in I}\mathcal{I}_i)(\mu_1) \wedge \mu_1 \in (\bigcap_{i\in I}\mathcal{I}_i)(\mu) \wedge \\ \mathsf{accept}\ y^{\ell_2} \in (\bigcap_{i\in I}\mathcal{I}_i)(\mu_2) \wedge \mu_2 \in (\bigcap_{i\in I}\mathcal{I}_i)(\mu) \wedge \\ \langle \bigcap_{i\in I}\mathcal{R}_i \rangle(x) \cap \langle \bigcap_{i\in I}\mathcal{R}_i \rangle(y) \neq \emptyset \wedge (\ell_1,\ell_2) \in \mathsf{CP}_\star\end{array}$$

It then follows that $\mu_1 \in \mathcal{I}_i(\mu_2) \wedge (\ell_1,\ell_2) \in \mathcal{F}_i$ for all $i \in I$, and hence that $\mu_1 \in (\bigcap_{i\in I}\mathcal{I}_i)(\mu_2) \wedge (\ell_1,\ell_2) \in (\bigcap_{i\in I}\mathcal{F}_i)$. This concludes the case.

The remaining cases are similar.

Case $\mathsf{rec}\,\mathrm{X}.\,P$**:** follows from the induction hypothesis.

Case X**:** Holds vacuously.

□

```
INPUT :    a Flow Logic $\mathcal{FL}$(BioAmbients, ALFP) and
           a BioAmbients process $P_\star$.
OUTPUT :   an ALFP formula $\varphi$ such that $(\mathcal{I},\mathcal{R},\mathcal{F}) \models \varphi \Leftrightarrow (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\top} P_\star$.
METHOD :   Set $\varphi := (\bigwedge\{\mathsf{CP}_\star(\ell_1,\ell_2) \mid (\ell_1,\ell_2) \in \mathsf{CP}_\star\}) \wedge$
                       $(\bigwedge\{\mathbf{C}(x) \mid x \in \mathbf{C}\}) \wedge$
                       $\mathsf{complete}\mathcal{R} \wedge$
                       $(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\top} P$
           while $\varphi$ contains $(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P'$
             and there is a rule $\alpha$ iff $\beta$ in $\mathcal{FL}$
                  and a substitution $\theta$
             such that $\theta\alpha = (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P'$
            do replace $(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P'$ with $\theta\beta$ in $\varphi$.
```

Table 6.6: 0CFA constraint generation algorithm.

Thus, trivially, $\bigsqcap\emptyset = (\top_{\mathcal{I}}, \top_{\mathcal{R}}, \top_{\mathcal{F}})$, is always an acceptable analysis estimate (the worst), and there is always a best estimate $(\mathcal{I}_\star, \mathcal{R}_\star, \mathcal{F}_\star) = \bigsqcap\{(\mathcal{I},\mathcal{R},\mathcal{F}) \mid (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P_\star\}$. As outlined in Section 4.3, we compute $(\mathcal{I}_\star, \mathcal{R}_\star, \mathcal{F}_\star)$ for a given $P_\star$ by first deriving an ALFP clause, φ, as shown in Table 6.6 and then computing the least model using *the Succinct Solver Suite* [NNS+04, NNS02], which then guarantees polynomial complexity.

6.3 CASE: Analysing the LDL Degradation Pathway

In order to investigate our model of the LDL degradation pathway (3.3) we shall now subject it to the 0CFA. The resulting $(\mathcal{I},\mathcal{R},\mathcal{F})$ is only included in Appendix B.1, but the corresponding ambient role containment hierarchy $(\mathcal{I})$ is shown graphically in Fig. 6.3.

Basically, the analysis results indicate that cholesterol (CH) might occur, in the lumen of every modelled compartment. This is in sharp contrast to the fact that CH can only really be released in the lumen of the lysosome ($LYSO$). Thus, the result is almost maximally un-informative. Due to the nature of over-approximation, however, we cannot guarantee that CH may actually reach the lumens of the indicated compartments.

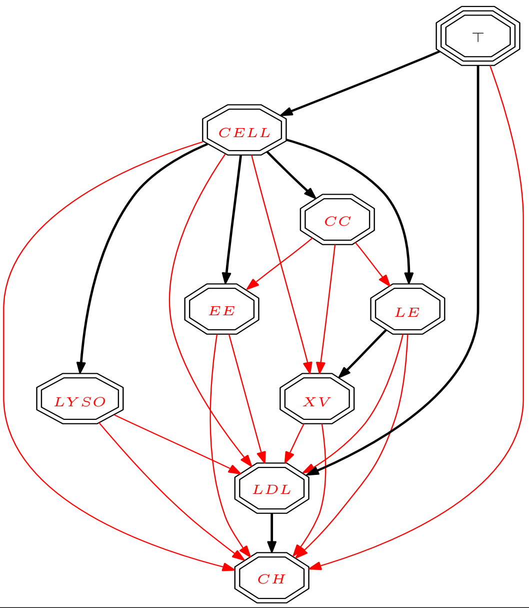


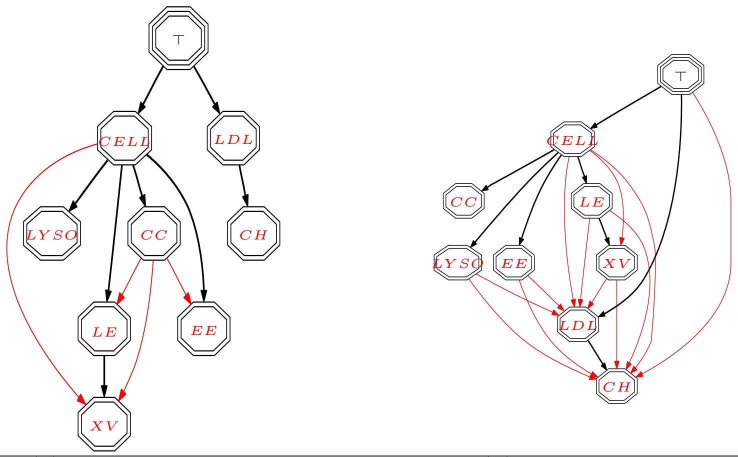
Figure 6.3: 0CFA applied to an LDL pathway with normal receptors. The special $\top$ node with triple borders denotes the super-environment. The remaining (double bordered) nodes represent ambients roles and the edges represent the containment relation $\mathcal{I}$. The bold black edges represent the system in its initial state. The thinner red edges account for the dynamic evolution of the system.

6.3.1 Related Diseases

As mentioned in Section 2.2.3 *familial hypercholesterolemia* is caused by defects in the genetic coding of the LDL receptor proteins.

When such a defect affects the exo-plasmic domain of the protein the receptor is no longer able to bind an LDL particle. As mentioned in Section 3.3 this corresponds to the model included in Appendix A.2. The corresponding 0CFA result, $(\mathcal{I}, \mathcal{R}, \mathcal{F})$, is reported in Appendix B.1.2, and the $\mathcal{I}$ relation, which is shown graphically in Fig. 6.4(a), reveals that the cell can no longer internalise LDL particles. It still internalises early endosomes (EE). Empty endosomes may occur inside the clathrin coat (CC) within the $CELL$, but in this case they carry no LDL cargo.

When the defect affects the intra-cellular part of the protein the receptor can bind, but not internalise, LDL particles. This corresponds to the model included in Appendix A.2. Unfortunately, the corresponding $\mathcal{I}$ relation, presented verba-



(a) extra-cellular defects (b) intra-cellular defects

Figure 6.4: 0CFA analysis of defect receptor LDL pathway models.

tim in Appendix B.1.2 and shown graphically in Fig. 6.4(b), barely shows the effect. In most aspects the graph is completely similar to the graph in Fig. 6.3, which represents the normal receptor model. However, it conclusively shows that the *EE* can never enter the *CC*. As this is a prerequisite of entering the *CELL* we conclude that, in fact, the *LDL* cannot enter. But how do we then explain the remaining spurious edges? Well, they occur due to a conspiracy of two factors:

- The aforementioned modelling artifact (Section 3.3) allows *EE* to enter *CELL* before entering the *CC*.
- The analysis is flow-insensitive, hence, the behaviour of the *EE* is analysed identically regardless of the reactions that have passed before. Or, in other words, the analysis does not care if the *EE* has passed through the *CC* or not.

Still, these results are slightly more encouraging than the normal receptor result would initially have us believe. In these particular cases the 0CFA is, just, precise enough to indicate that the system, as modelled here, cannot perform its normal function if severely perturbed.

6.4 Concluding Remarks

The 0CFA presented in this chapter extends a tradition of CFAs for ambient calculi that began with [HJNN99, NNHJ99]. In the biological branch of this tradition the work reported here, largely that of [NNP04], emerged as an evolution of the 0CFA presented in [NNPdR07].

The 0CFA presented in this chapter differs from that presented in [NNPdR07] in two major respects:

In [NNPdR07] communication causes updates in both $\mathcal{I}$ and $\mathcal{R}$. In this dissertation, however, we keep the binding information strictly confined to $\mathcal{R}$, store only representatives in $\mathcal{I}$, and then expand these representatives "on the fly". This approach optimises with respect to space consumption, which has proven a sensible strategy when using the Succinct Solver [NNS02]. In this context, however, we must expand the $\mathcal{I}$ relation into $(\mathcal{I}@\mathcal{R}, \mathcal{R})$ when formulating the correctness result with respect to a substitution based reaction semantics.

In [NNPdR07] the closure conditions may match and fire any two capabilities that might occur in the right positions of the ambient hierarchy. Here we take inspiration from [BDPZ03] and use an auxiliary relation $\mathsf{CP}_\star$ to impose a restriction; only capabilities that might be concurrently possible may be matched.

Furthermore, the 0CFA presented here also differs from the work originally published in [NNP04] in a couple of respects:

In [NNP04] a relation similar to $\mathsf{CP}_\star$ is used to control the application of the closure conditions. At the time, however, it had not yet been realised that, in the context of general recursion, $\mathsf{CP}_\star$ is really a fixed point property. Here we rectify this mistake by using an indexed hierarchy of functions, $\mathsf{cap}_{\Gamma_{\mathsf{cap}}}()$ and $\mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}()$, to compute the necessary information.

In [NNP04] process identifiers are allowed to occur within ambient boundaries. Hence, the developments of [NNP04] include a general mechanism for ensuring that the corresponding body is analysed in all relevant contexts that may arise dynamically. For the purpose of this dissertation we see no biological relevance in allowing free process identifiers within ambient boundaries and, hence, this development is not included.

There has previously been positive comments [NNPdR07][NNP04] regarding the use of 0CFA for biological applications. However, the results of this chapter seem to indicate, quite conclusively, that the presented 0CFA is too weak. The analysis results of both the running example and, in particular, the LDL degradation pathway model exhibits poor precision, which emerges as spurious edges in the corresponding graphical representations.

Reflecting upon this, it seems obvious that the observed imprecision owes to lack of context and, to a lesser degree, flow information. This conclusion is what originally motivated the context sensitive Control Flow Analysis presented in the following chapter.

Chapter 7

Context Sensitive Control Flow Analysis

"From naive simplicity we arrive at more profound simplicity."
— Albert Schweitzer

This chapter presents a *context sensitive* or *poly-variant Control Flow Analysis* (*2CFA*) for the BioAmbients language and evaluates it by analysing the LDL degradation pathway of Section 3.3. Parts of the material was previously published in [PNN06b].

The analysis uses two levels of context information and, like the 0CFA of the previous chapter, it is defined as a Flow-Logic

$$2\text{CFA} = \mathcal{FL}(\text{BioAmbients}, \textsc{alfp})$$

that keeps track of the contents of ambients and their abilities for participating in various interactions and, thus, shares many characteristics with previous analyses of Mobile Ambients [NNH02, NNB04, LM01, NNPdR07, NNP04]

The analysis is structurally similar to the 0CFA, but it adds several novel elements in order to increase precision:

First, the Control Flow Analysis is *context sensitive*, i.e., it approximates the containment contexts of ambients, capabilities, and bindings and is thereby able to differentiate between interactions that are possible in different settings. It has been found that *two* enclosing ambient roles suffices for analysing fixed-structure cellular models with simple intruders [PNN05]. This strikes a balance between the classical approaches that completely ignore context [NNH02, NNPdR07] and the more complex shape analysis of [NN00].

Second, the use of contexts means that extensive copying is delegated to the closure conditions, and this offers multiple handles for controlling the precision:

Whenever two capabilities might match to cause an ambient movement then the moving ambient, the *relevant* parts of its contents, and the *relevant* associated variable bindings might also occur in some 'new' context. Here 'relevant' refers to those that do not disappear due to non-deterministic choice when the movement occurs. In the case of variable bindings this notion of relevance is formally captured by an auxiliary relation $\mathsf{RV}_\star$, that safely approximates the *relevant variables*, and is used to control the copying of variable bindings within $\mathcal{R}$ that occurs in conjunction with ambient movement. And, in the case of ambient contents, by an auxiliary relation $\mathsf{RPA}_\star$ that safely approximates the *relevant prefixes and ambients*, and is used to control the copying of capabilities and prefixes within $\mathcal{I}$ that occurs in conjunction with ambient movement.

Furthermore, whenever two capabilities might match to cause a communication then all names that might be communicated might become bound, both in the immediate context of the receiving capability, M^ℓ, and in all *relevant* sub-contexts. Here the relevant sub-contexts are the ambients that lie within the static scope of the variable bound by M^ℓ. This notion of relevance is formally captured by an auxiliary relation $\mathsf{SCP}_\star$, that safely approximates the *spatial shape of static scope*, and is used to control the propagation of name bindings into sub-contexts.

The chapter is in six sections. First, in Section 7.1, we define the auxiliary relation $\mathsf{SCP}_\star$, and show that, for each variable, it constitutes a safe approximation to the spatial shape of the corresponding static scope. In Section 7.2 we define the auxiliary relation $\mathsf{RV}_\star$, and show that it constitutes a safe approximation to the set of relevant variables of each capability. Similarly, in Section 7.3, we define the auxiliary relation $\mathsf{RPA}_\star$, and show that it constitutes a safe approximation to the set of relevant prefixes and ambients of each capability, Then, in Section 7.4, we specify the 2CFA analysis as a Flow Logic, show that it is correct, and how to compute it. In Section 7.5 we apply the 2CFA to the LDL degradation pathway in order to both evaluate the analysis and obtain information about the LDL pathway. Finally, in Section 7.6 we summarise our findings.

$$
\begin{aligned}
\mathsf{amb}_{\Gamma_{\mathsf{amb}}}((n)\,P) &= \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P)\\
\mathsf{amb}_{\Gamma_{\mathsf{amb}}}([\![\, P \,]\!]^{\mu}) &= \{\mu\} \cup \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P)\\
\mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P \,|\, Q) &= \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P) \cup \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(Q)\\
\mathsf{amb}_{\Gamma_{\mathsf{amb}}}(\sum_{i\in I} M_i^{\ell_i} \,.\, P_i) &= \bigcup_{i\in I} \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P_i)\\
\mathsf{amb}_{\Gamma_{\mathsf{amb}}}(\mathsf{rec}\, \mathrm{X}.\, P) &= \mathsf{amb}_{\Gamma_{\mathsf{amb}}[X\mapsto\emptyset]}(P)\\
\mathsf{amb}_{\Gamma_{\mathsf{amb}}}(X) &= \Gamma_{\mathsf{amb}}(X)
\end{aligned}
$$

Table 7.1: Ambient roles, $\mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P)$, occurring in a process, P.

7.1 The Spatial Shape of Static Scope

When defining the 2CFA in Section 7.4 we shall find it useful to have an a priori given estimate of the *spatial shape of static scopes*, i.e., a relation that associates every variable with the ambients where it might be in scope.

In order to specify the required information we shall first define the set of *occurring ambients*, $\mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P)$, of a process P. The set is defined in terms of the recursive function of Table 7.1 and, as usual, it is parameterised on an environment, $\Gamma_{\mathsf{amb}} : \mathbf{Pid} \to \mathcal{P}(\mathbf{Role})$, used to associate process identifiers with sets of ambient roles. The function performs a straightforward recursive descent in order to simply accumulate the set of occurring ambient roles. Consequently, the main case is that for the ambient boundary.

In this context we capture the spatial shape of the static scopes, $\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P)$, that occur in a process, P, by associating each occurring bound variable with the set of ambient roles that occur in its static scope. This is captured by the function shown in Table 7.2, which is parameterised on the aforementioned Γ_{amb} environment as well as another environment, $\Delta_{\mathsf{SCP}} : \mathbf{Pid} \to \mathcal{P}(\mathbf{V} \times \mathbf{Role})$, used to associate process identifiers with spatial shapes of static scopes. The interesting case is that of summation, where each variable occurring bound in a prefix, $M_i^{\ell_i}$, is associated with the set of ambient roles occurring in the corresponding continuation process P_i.

Convention 7.1 When P is known to be identifier closed we write $\mathsf{SCP}(P)$, rather than $\mathsf{SCP}_{[\,],[\,]}(P)$, to denote the spatial shape of static scope of P. When furthermore the subject of approximation is an initial program, $P_\star$, we write $\mathsf{SCP}_\star$ for $\mathsf{SCP}(P_\star)$. ■

$$
\begin{aligned}
\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}((n)\,P) &= \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P)\\
\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}([\,P\,]^{\mu}) &= \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P)\\
\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P \mid Q) &= \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P) \cup \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(Q)\\
\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(\sum_{i\in I} M_i^{\ell_i}\,.\,P_i) &= \bigcup_{i\in I}(\ \ (\mathsf{bn}(M_i) \times \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P_i)) \cup \\
&\qquad\quad \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P_i)\ )\\
\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(\mathsf{rec}\,\mathrm{X}.\,P) &= \mathsf{SCP}_{\Gamma_{\mathsf{amb}}[X\mapsto \mathsf{amb}_{\Gamma_{\mathsf{amb}}[X\mapsto\emptyset]}(P)],\Delta_{\mathsf{SCP}}[X\mapsto\emptyset]}(P)\\
\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(X) &= \Delta_{\mathsf{SCP}}(X)
\end{aligned}
$$

Table 7.2: Static scopes, $\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P)$, of a process, P.

Soundness of $\mathsf{SCP}_\star$. We shall now show that for any $P_\star$ the corresponding $\mathsf{SCP}_\star$ constitutes a safe over-approximation, i.e.,

$$\text{if } P_\star \xrightarrow{\tilde{L}}{}^{\star} P \text{ then } \mathsf{SCP}_\star \supseteq \mathsf{SCP}(P)$$

The proof involves a number of minor results.

Fact 7.2 Assume $Q = \mathsf{rec}\,\mathrm{X}.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

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

Proof The result follows by structural induction on P. □

Facts 7.3 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

1. If $P \Rrightarrow Q$ then $\mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P) \supseteq \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(Q)$.
2. If $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P) \supseteq \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(Q)$.

Proof The proof of *(1)* is by induction on the shape of the proof tree establishing $P \Rrightarrow Q$. We use Fact 7.2 in the case of H-UREC. The remaining axioms are straightforward and the rules all follow from the induction hypothesis, and, in the case of H-TRAN, the transitivity of $\supseteq$.

The proof of *(2)* is by induction on the shape of the proof tree establishing

$P \xrightarrow{\tilde{\ell}} Q$. The axioms follow by calculation. The rules follow by the induction hypothesis, using *(1)* in the case of R-AUX. □

Fact 7.4 Assume $Q = \mathsf{rec}\, X.\, Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P[^{Q}/_{X}]) = \begin{cases} \mathsf{SCP}_{\Gamma'_{\mathsf{amb}},\Delta'_{\mathsf{SCP}}}(P) \cup \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(Q) & \text{if } X \in \mathsf{fpi}(P) \\ \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P) & \text{otherwise} \end{cases}$$

where $\Gamma'_{\mathsf{amb}} = \Gamma_{\mathsf{amb}}[X \mapsto \mathsf{amb}_{\Gamma_{\mathsf{amb}}[X\mapsto\emptyset]}(Q)]$ and $\Delta'_{\mathsf{SCP}} = \Delta_{\mathsf{SCP}}[X \mapsto \emptyset]$.

Proof The proof proceeds by structural induction in P. The base case and most other cases are trivial. The most interesting case is that of summation where, for each summand $M^{\ell}\,.\,P$, if $X \in \mathsf{fpi}(M^{\ell}\,.\,P)$, then:

$$\begin{aligned} \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}((M^{\ell}\,.\,P)[^{Q}/_{X}]) &= \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(M^{\ell}\,.\,P[^{Q}/_{X}]) \\ &= \mathsf{bn}(M^{\ell}) \times \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P[^{Q}/_{X}]) \cup \\ &\quad\ \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P[^{Q}/_{X}]) \\ \text{(by ind. hyp.)} &= \mathsf{bn}(M^{\ell}) \times \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P[^{Q}/_{X}]) \cup \\ &\qquad \mathsf{SCP}_{\Gamma_{\mathsf{amb}}[X\mapsto\mathsf{amb}_{\Gamma_{\mathsf{amb}}[X\mapsto\emptyset]}(Q)],\Delta_{\mathsf{SCP}}[X\mapsto\emptyset]}(P) \cup \\ &\qquad \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(Q) \\ \text{(by Fact 7.2 \& def. SCP)} &= \mathsf{SCP}_{\Gamma_{\mathsf{amb}}[X\mapsto\mathsf{amb}_{\Gamma_{\mathsf{amb}}[X\mapsto\emptyset]}(Q)],\Delta_{\mathsf{SCP}}[X\mapsto\emptyset]}(M^{\ell}\,.\,P) \cup \\ &\qquad \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(Q). \end{aligned}$$

In the case of the recursive process we make similar use of Fact 7.2. □

Facts 7.5 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

1. If $P \Rrightarrow Q$ then $\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P) \supseteq \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(Q)$.
2. If $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(P) \supseteq \mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}(Q)$.

Proof The proof of *(1)* follows by induction on the shape of the proof tree establishing $P \Rrightarrow Q$. Most axioms are straightforward. However, in the case of H-UREC we use Fact 7.4. The rules all follow from the induction hypothesis, and, in the case of H-TRAN, transitivity of $\supseteq$.

The proof of *(2)* proceeds by induction on the shape of the prooftree establishing $P \xrightarrow{\tilde{\ell}} Q$. Again, the axioms follow by straightforward calculation. The rules mostly follow directly from the induction hypothesis, but, in the case of R-AUX, also depends on *(1)*. □

Corollary 7.6 (Subject reduction)
If $\mathsf{PRG}_{\mathbf{C}}(P)$ and $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{SCP}(P) \supseteq \mathsf{SCP}(Q)$. □

Finally, the sought result emerges as a special case of the following lemma:

Lemma 7.7 (Semantic correctness) Assume that $P_\star$ is a well-formed initial program then:

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

Proof The result follows by induction on the length of the derivation sequence $P_\star \xrightarrow{\tilde{L}}{}^{\star} P$. The base case is given by Corollary 7.6 and the inductive step follows from the induction hypothesis, Corollary 7.6, and Corollary 5.13. □

7.2 Relevant Variables

We shall also find use for a safe estimate of the variable bindings that remain relevant after a reaction, i.e., a relation associating each capability with those variables that do not disappear due to non-deterministic choice when the capability engages in reaction.

In order to specify this information we first need to formalise the *free variables*, $\mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P)$, of a process P. We capture this by the function shown in Table 7.3, which, in contrast to the free names function used by the semantics, only considers variables. The associated environment, $\Gamma_{\mathsf{fv}} : \mathbf{Pid} \to \mathcal{P}(\mathbf{V})$, associates process identifiers with sets of free variables. The interesting information is collected in the case of summation. The canonical version of each *free* name that occurs in a prefix $M_i^{\ell_i}$, but not in the set of constants $\mathbf{C}$, is collected. We are computing an over-approximation and therefore the information synthesised from various sub-expressions is combined by taking the union of the sets.

In this context we capture the *relevant variables*, $\mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(P)$, occurring in a process, P, by associating each occurring capability label, ℓ, with the set of variables that may occur either in parallel to, or sequentially after, ℓ. The information is computed as shown in Table 7.4. Here Γ_{fv} is the environment used by $\mathsf{fv}_{\Gamma_{\mathsf{fv}}}()$, Γ_{cap} the environment used by $\mathsf{cap}_{\Gamma_{\mathsf{cap}}}()$ (Section 6.1), and $\Delta_{\mathsf{RV}} : \mathbf{Pid} \to \mathcal{P}(\mathbf{Lab} \times \mathbf{V})$ is an environment that associates process identifiers with relevant names estimates. We shall briefly examine the interesting cases:

$$
\begin{aligned}
\mathsf{fv}_{\Gamma_{\mathsf{fv}}}((n)\,P) &= \mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P)\\
\mathsf{fv}_{\Gamma_{\mathsf{fv}}}([\,P\,]^{\mu}) &= \mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P)\\
\mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P \mid Q) &= \mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P) \cup \mathsf{fv}_{\Gamma_{\mathsf{fv}}}(Q)\\
\mathsf{fv}_{\Gamma_{\mathsf{fv}}}(\sum_{i\in I} M_i^{\ell_i}\,.\,P_i) &= \bigcup_{i\in I}((\lfloor \mathsf{fn}(M_i)\rfloor \setminus \mathbf{C}) \cup (\lfloor \mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P_i)\rfloor \setminus \lfloor \mathsf{bn}(M_i)\rfloor))\\
\mathsf{fv}_{\Gamma_{\mathsf{fv}}}(\mathsf{rec}\,\mathrm{X}.\,P) &= \mathsf{fv}_{\Gamma_{\mathsf{fv}}[X\mapsto\emptyset]}(P)\\
\mathsf{fv}_{\Gamma_{\mathsf{fv}}}(X) &= \Gamma_{\mathsf{fv}}(X)
\end{aligned}
$$

Table 7.3: Free variables, $\mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P)$, of a process P.

$$
\begin{aligned}
\mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}((n)\,P) &= \mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(P)\\
\mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}([\,P\,]^{\mu}) &= \mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(P)\\
\mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(P \mid Q) &= \mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(P) \cup \mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(Q) \cup\\
&\quad (\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P) \times \mathsf{fv}_{\Gamma_{\mathsf{fv}}}(Q)) \cup (\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(Q) \times \mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P))\\
\mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(\sum_{i\in I} M_i^{\ell_i}\,.\,P_i) &= \bigcup_{i\in I}((\{\ell_i\} \times \mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P_i)) \cup \mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(P_i))\\
\mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(\mathsf{rec}\,\mathrm{X}.\,P) &= \mathsf{RV}_{\substack{\Gamma_{\mathsf{fv}}[X\mapsto \mathsf{fv}_{\Gamma_{\mathsf{fv}}[X\mapsto\emptyset]}(P)],\\ \Gamma_{\mathsf{cap}}[X\mapsto \mathsf{cap}_{\Gamma_{\mathsf{cap}}[X\mapsto\emptyset]}(P)],\\ \Delta_{\mathsf{RV}}[X\mapsto\emptyset]}}(P)\\
\mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(X) &= \Delta_{\mathsf{RV}}(X)
\end{aligned}
$$

Table 7.4: Relevant variables, $\mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(P)$, of a process, P.

- In the case of parallel composition, $P \mid Q$, the union of two direct products is recorded; the product of all capabilities of P (i.e., $\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P)$) and all free (canonical) variables of Q (i.e., $\mathsf{fv}_{\Gamma_{\mathsf{fv}}}(Q)$) – and vice versa.
- For each summand $M_i^{\ell_i}\,.\,P_i$ of a summation the label ℓ_i is associated with the free variables of the continuation P_i (i.e., $\mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P_i)$).
- In the case of a recursive process, $\mathsf{rec}\,\mathrm{X}.\,P$, Γ_{cap} and Γ_{fv} are updated in order to correctly associate X with, respectively, the set of labelled capabilities and the set of free variables of P.

The function computes an over-approximation; hence the information emerging from sub-expressions is combined by taking the union of the emerging sets.

Example 7.8 For the running example P_{eat} we get

$$\mathsf{RV}_{P_{\mathsf{eat}}} = \{(\ell_2, rl)\}$$

which in fact indicates that no names are relevant to any ambient movement. □

Convention 7.9 When P is known to be identifier closed we write $\mathsf{RV}(P)$, rather than $\mathsf{RV}_{[\,],[\,],[\,]}(P)$, to denote the relevant variables of P. When furthermore the subject of approximation is an initial program, $P_\star$, we write $\mathsf{RV}_\star$ for $\mathsf{RV}(P_\star)$. ■

Soundness of $\mathsf{RV}_\star$. We shall now show that for any $P_\star$ the corresponding $\mathsf{RV}_\star$ constitutes a safe over-approximation, i.e.,

$$\text{if } P_\star \xrightarrow{\tilde{L}}{}^\star P \text{ then } \mathsf{RV}_\star \supseteq \mathsf{RV}(P)$$

meaning that the relation $\mathsf{RV}_\star$ correctly relates the label of each capability, M^ℓ, occurring in P to all variables in P that might be relevant, i.e., those that do not disappear because of non-deterministic choice when M^ℓ engages in reaction.

The proof involves a number of minor results.

Fact 7.10 Assume $Q = \mathsf{rec}\,\mathrm{X}.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

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

Proof The result follows by induction in the structure of P. In the case of summations we rely on the well-formedness of Q, and the fact that P is a subprocess of Q, to ensure that the bound variables of P cannot capture the free variables of Q. □

Facts 7.11 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

1. If $P \Rrightarrow Q$ then $\mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P) \supseteq \mathsf{fv}_{\Gamma_{\mathsf{fv}}}(Q)$.
2. If $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{fv}_{\Gamma_{\mathsf{fv}}}(P) \supseteq \mathsf{fv}_{\Gamma_{\mathsf{fv}}}(Q)$.

Proof The proof of *(1)* is by induction on the shape of the proof tree establishing $P \Rrightarrow Q$. The base case is straightforward. In the case of H-UREC we use Fact 7.10. The remaining cases follow by the induction hypothesis.

The proof of *(2)* is by induction on the shape of the proof tree establishing $P \xrightarrow{\tilde{\ell}} Q$. In the cases of movement and communication axioms the result follows by simple calculation. In the case of R-AUX we use *(1)*. The remaining cases follow by the induction hypothesis. □

Fact 7.12 Assume $Q = \mathsf{rec}\,\mathrm{X}.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$\mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(P[^{Q}/_{X}]) = \begin{cases} \mathsf{RV}_{\Gamma'_{\mathsf{fv}},\Gamma'_{\mathsf{cap}},\Delta_{\mathsf{RV}}[X\mapsto\emptyset]}(P) \cup \\ \qquad \mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(Q) & \text{if } X \in \mathsf{fpi}(P) \\ \mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(P) & \text{otherwise} \end{cases}$$

where $\Gamma'_{\mathsf{fv}} = \Gamma_{\mathsf{fv}}[X \mapsto \mathsf{fv}_{\Gamma_{\mathsf{fv}}[X\mapsto\emptyset]}(Q)]$ and $\Gamma'_{\mathsf{cap}} = \Gamma_{\mathsf{cap}}[X \mapsto \mathsf{cap}_{\Gamma_{\mathsf{cap}}[X\mapsto\emptyset]}(Q)]$.

Proof The proof proceeds by structural induction on P. The base case is simple. In the cases of parallel composition and recursive processes we rely on Fact 6.3, Fact 7.10, and the induction hypothesis. In the case of summation we only need Fact 7.10 and the induction hypothesis. The remaining cases follow by the induction hypothesis. □

Facts 7.13 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

1. If $P \Rrightarrow Q$ then $\mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(P) \supseteq \mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(Q)$.

2. If $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(P) \supseteq \mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}(Q)$.

Proof The proof of *(1)* follows by induction on the shape of the proof tree establishing $P \Rrightarrow Q$. Most base cases are straightforward. In the case of H-UREC, however, we rely on Fact 7.12. In the case of H-CPAR we use Fact 6.4-*(1)*, Fact 7.11-*(1)*, and the induction hypothesis. In the case of H-CSUM we use 7.11-*(1)* and the induction hypothesis. The remaining rules follow by the induction hypothesis.

The proof of *(2)* proceeds by induction on the shape of the prooftree establishing $P \xrightarrow{\tilde{\ell}} Q$. The movement axioms follow by calculation. The communication axioms likewise, but here we use the observation that e.g.

$$\mathsf{fv}_{\Gamma_{\mathsf{fn}}}(n!\{n\}^{\ell}\,.\,Q) = \mathsf{fv}_{\Gamma_{\mathsf{fn}}}(Q[^{m}/_{p}])$$

for all $n, m \in \mathbf{C}$. In the case of R-PAR we use Fact 6.4-*(2)* and Fact 7.11-*(2)*. In the case of R-AUX we use *(1)*. The remaining rules follow by the induction hypothesis. □

Corollary 7.14 (Subject reduction)
If $\mathsf{PRG}_{\mathbf{C}}(P)$ and $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{RV}(P) \supseteq \mathsf{RV}(Q)$. □

Finally, the sought result emerges as a special case of the following lemma:

Lemma 7.15 (Semantic correctness) Assume that $P_\star$ is a well-formed initial program then:

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

Proof The result follows by induction on the length of the derivation sequence $P_\star \xrightarrow{\tilde{L}}{}^\star P$. The base case is given by Corollary 7.14 and the inductive step follows from the induction hypothesis, Corollary 7.14, and Corollary 5.13. □

7.3 Relevant Prefixes and Ambient Roles

As for variables we shall also be interested in the relevance of capabilities and ambient roles when we specify the 2CFA. Thus we shall now specify a safe estimate of the capabilities and ambients that remain relevant after a transition, i.e., a relation associating each capability prefix M^ℓ with those capabilities and ambients that do not disappear due to non-deterministic choice when M^ℓ engages in reaction.

Thus the intuition behind the approximation is similar to that of the relevant variables approximation and, as we shall see, the method is also similar. The *relevant prefixes and ambients*, $\mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(P)$, of a process, P, is the set specified by the function of Table 7.5. The environments Γ_{cap} and Γ_{amb} are as before, but $\Delta_{\mathsf{RPA}} : \mathbf{Pid} \to \mathcal{P}(\mathbf{Lab} \times \mathbf{Role})$ can be used to associate process identifiers to sets of relevant prefixes and ambients. and we shall briefly examine the intersting cases:

- In the case of parallel composition, $P \mid Q$, the union of two direct products is recorded; the product of all capability labels of P (i.e., $\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P)$) and all occurring prefixes and ambients of Q (i.e., $\mathsf{amb}_{\Gamma_{\mathsf{amb}}}(Q) \cup \mathsf{cap}_{\Gamma_{\mathsf{cap}}}(Q)$) – and vice versa.
- For each summand $M_i^{\ell_i} \,.\, P_i$ of a summation the label ℓ_i is paired with the set of prefixes and ambients occurring in the continuation P_i.

$$\begin{aligned}
\mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}((n)\,P) &= \mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(P)\\
\mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}([\![\, P \,]\!]^{\mu}) &= \mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(P)\\
\mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(P \mid Q) &= \mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(P) \cup \mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(Q) \cup\\
&\quad (\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P) \times (\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(Q) \cup \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(Q))) \cup\\
&\quad (\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(Q) \times (\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P) \cup \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P)))\\
\mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(\sum_{i\in I} M_i^{\ell_i}\,.\,P_i) &= \bigcup_{i\in I}(\ (\{\ell_i\} \times (\mathsf{cap}_{\Gamma_{\mathsf{cap}}}(P_i) \cup \mathsf{amb}_{\Gamma_{\mathsf{amb}}}(P_i))) \cup\\
&\quad \mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(P_i)\ )\\
\mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(\mathsf{rec}\,\mathrm{X}.\,P) &= \mathsf{RPA}_{\substack{\Gamma_{\mathsf{cap}}[X\mapsto \mathsf{cap}_{\Gamma_{\mathsf{cap}}[X\mapsto\emptyset]}(P)],\\ \Gamma_{\mathsf{amb}}[X\mapsto \mathsf{amb}_{\Gamma_{\mathsf{amb}}[X\mapsto\emptyset]}(P)],\\ \Delta_{\mathsf{RPA}}[X\mapsto\emptyset]}}(P)\\
\mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(X) &= \Delta_{\mathsf{RPA}}(X)
\end{aligned}$$

Table 7.5: Relevant prefixes and ambients, $\mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(P)$, of P.

- For a recursive process, $\mathsf{rec}\,\mathrm{X}.\,P$, Γ_{cap} and Γ_{amb} are updated in order to correctly associate X with the set of capability labels and ambient roles, respectively, of P.

Again, as the function defines an over-approximation it combines the information emerging from sub-expression by taking the union of the computed sets.

Convention 7.16 When P is known to be identifier closed we write $\mathsf{RPA}(P)$, rather than $\mathsf{RPA}_{[],[],[]}(P)$, to denote the relevant prefixes and ambients of P. When furthermore the subject of approximation is an initial program, $P_\star$, we write $\mathsf{RPA}_\star$ for $\mathsf{RPA}(P_\star)$. ■

Soundness of $\mathsf{RPA}_\star$. We shall now show that for any $P_\star$ the corresponding $\mathsf{RPA}_\star$ constitutes a safe over-approximation, i.e.,

$$\text{if } P_\star \xrightarrow{\tilde{L}}{}^{\star} P \text{ then } \mathsf{RPA}_\star \supseteq \mathsf{RPA}(P)$$

meaning that the relation $\mathsf{RPA}_\star$ correctly relates the label of each capability occurring in P to the labels of all capabilities and the roles of all ambients in P that might be relevant to it, i.e., names that do not disappear because of non-deterministic choice. The proof involves a few minor results.

Fact 7.17 Assume $Q = \mathsf{rec}\,\mathrm{X}.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$\mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(P[{}^{Q}/_{X}]) = \begin{cases} \mathsf{RPA}_{\Gamma'_{\mathsf{cap}},\Gamma'_{\mathsf{amb}},\Delta_{\mathsf{RPA}}[X\mapsto\emptyset]}(P) \cup \\ \qquad \mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(Q) & \text{if } X \in \mathsf{fpi}(P) \\ \mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(P) & \text{otherwise} \end{cases}$$

where $\Gamma'_{\mathsf{cap}} = \Gamma_{\mathsf{cap}}[X \mapsto \mathsf{cap}_{\Gamma_{\mathsf{cap}}[X\mapsto\emptyset]}(P)]$ and $\Gamma'_{\mathsf{amb}} = \Gamma_{\mathsf{amb}}[X \mapsto \mathsf{amb}_{\Gamma_{\mathsf{amb}}[X\mapsto\emptyset]}(P)]$.

Proof The proof proceeds by structural induction in P.The base case is straightforward. In the case of parallel composition, summation, and recursive processes we utilise the induction hypothesis in conjunction with Fact 6.3 and 7.2. The remaining cases follow by the induction hypothesis. □

Facts 7.18 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

1. If $P \Rrightarrow Q$ then $\mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(P) \supseteq \mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(Q)$.

2. If $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(P) \supseteq \mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}(Q)$.

Proof The proof of *(1)* follows by induction on the shape of the proof tree establishing $P \Rrightarrow Q$. Most of the axioms are straightforward. In the case of H-UREC, however, we use Fact 7.17. In the case of H-CPAR and H-CSUM we use the induction hypothesis in conjunction with Fact 6.4-*(1)* and Fact 7.3-*(1)*. The remaining rules follow from the induction hypothesis.

The proof of *(2)* proceeds by induction on the shape of the proof tree establishing $P \xrightarrow{\tilde{\ell}} Q$.The axioms all follow by toilsome calculation. In the case of R-PAR we use the induction hypothesis in conjunction with Fact 6.4-*(2)* and Fact 7.3-*(2)*. In case of R-AUX we use the induction hypothesis in conjunction with *(2)*. The remaining cases follow by the induction hypothesis. □

Corollary 7.19 (Subject reduction)
If $\mathsf{PRG}_{\mathbf{C}}(P)$ and $P \xrightarrow{\tilde{\ell}} Q$ then $\mathsf{RPA}(P) \supseteq \mathsf{RPA}(Q)$. □

Finally, the sought result emerges as a special case of the following lemma:

Lemma 7.20 (Semantic correctness) Assume that $P_\star$ is a well-formed initial program then:

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

Proof The result follows by induction on the length of the derivation sequence $P_\star \xrightarrow{\tilde{L}}{}^\star P$. The base case is given by Corollary 7.19 and the inductive step follows from the induction hypothesis, Corollary 7.19, and Corollary 5.13. □

7.4 Control Flow Analysis

Being both context and flow insensitive the 0CFA approach of the previous chapter leads to highly imprecise analysis estimates. The 2CFA analysis of this chapter is context sensitive: Whenever the analysis makes an entry into $\mathcal{I}$ or $\mathcal{R}$, corresponding to a semantic action that changes the system, it also records the k ambients enclosing the site of change. This can be done in relations $\mathcal{I} \subseteq \mathbf{Role^k} \times (\mathbf{Role} \cup (\mathbf{Cap} \times \mathbf{Lab}))$ and $\mathcal{R} \subseteq \mathbf{Role^k} \times \mathbf{Name} \times \mathbf{Name}$.

The parameter k is subject to conflicting interests and must be chosen with care:

- On the one hand, for the analysis to be precise, k must be large enough to locate all tracked entities uniquely in the ambient hierarchy of the model.
- On the other hand, for the analysis to be computationally efficient, k must not be too large.

In terms of nesting, the eukaryotic cell, shown schematically in Fig. 7.1, is the most complex entity of Molecular Biology. It contains a number of interior compartments called organelles. Some of these are nested structures with multiple compartments within compartments.

The analysis is intended to distinguish compartments only up to their roles; all lysosomes are the same. Thus, it is safe to assume that all roles are uniquely placed within the nesting hierarchy of the cell, and hence that only the nesting depth of ingested molecules are relevant to k.

No biological processes, except for certain cases of phagocytosis, i.e., the ingestion of large foreign bodies by macrophages, allow truly nested entities into the living cell. Some biological processes, though, can only be modelled in BioAmbients if non-nested entities are encoded as ambients; the running example, P_{eat}, and the LDL pathway model of Section 3.3 both rely on this type of modelling.

Encodings normally require two levels of context ($k = 2$). Three layer encodings

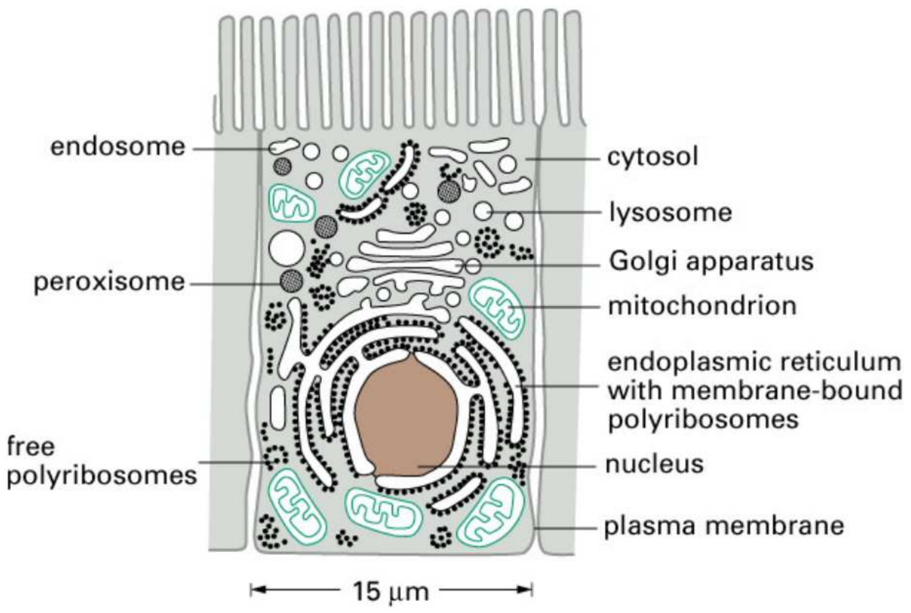


Figure 7.1: Schematic view of eukaryotic cell. Copyright 2002 from Molecular Biology of the Cell by Alberts et al [AJL$^+$02]. Reproduced by permission of Garland Science/Taylor & Francis LLC.

($k = 3$) seem contrived but may rarely be useful. Thus for the purpose of this dissertation it is assumed that $k = 3$ is sufficient to analyse any single cell system. We call the resulting analysis, which we shall specify in the following, a 2CFA because the 0CFA of the previous chapter, where $k = \frac{1}{2}$, is used as the baseline. Higher values of k may be required for the modelling of multicellular systems, because different cell types contain similar compartments and the aim is to distinguish between these. Of course the technical developments generalize straightforwardly to any k, but using a clever naming scheme when modelling might be a better option,

7.4.1 The Analysis Domain

Notation 7.21 In the following we shall write $\overset{\twoheadrightarrow}{\mu}$ to denote μ_{gp}, μ_p, μ and $\vec{\mu}$ to denote μ_p, μ. ■

The analysis specifies the following three components:

- A localised approximation of the relevant name bindings:
$$\mathcal{R} \subseteq \mathbf{Role^3} \times \mathbf{V} \times \mathbf{C}$$
where we write $n \in \mathcal{R}(\overrightarrow{\mu}, p)$ or $(\overrightarrow{\mu}, p, n) \in \mathcal{R}$ to assert the truth of predicate $\mathcal{R}(\overrightarrow{\mu}, p, n)$, i.e., that $\mathcal{R}$ records that the variable p may be bound to the name n in the context of $\overrightarrow{\mu}$.

- A localised approximation of the contents of ambients:
$$\mathcal{I} \subseteq \mathbf{Role^3} \times (\mathbf{Role} \cup (\mathbf{Cap} \times \mathbf{Lab}))$$
where we write $\mu' \in \mathcal{I}(\overrightarrow{\mu})$ or $(\overrightarrow{\mu}, \mu') \in \mathcal{I}$ (or, $M^\ell \in \mathcal{I}(\overrightarrow{\mu})$ or $(\overrightarrow{\mu}, M^\ell) \in \mathcal{I}$) to denote the truth of predicate $\mathcal{I}(\overrightarrow{\mu}, \mu')$ (or $\mathcal{I}(\overrightarrow{\mu}, M^\ell)$), i.e., that $\mathcal{I}$ records that an ambient of role μ' (or a prefix M^ℓ) may occur in the context of $\overrightarrow{\mu}$.

- An approximation of the pairs of capabilities that may react:
$$\mathcal{F} \subseteq \mathbf{Lab} \times \mathbf{Lab}$$
where we shall write $(\ell_1, \ell_2) \in \mathcal{F}$ to denote the truth of $\mathcal{F}(\ell_1, \ell_2)$, i.e. that $\mathcal{F}$ records that prefixes labelled ℓ_1 and ℓ_2 may react.

The domain of the 2CFA is then the direct product of those corresponding to the three components, which constitutes a complete lattice under the component-wise subset ordering.

7.4.2 The Acceptability Judgement

An *acceptability judgement* defines the set of acceptable 2CFA estimates. It takes the form
$$(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\mu_{gp}, \mu_p, \mu} P$$
and expresses that a 2CFA estimate $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ is acceptable for a (sub-)expression P (of $P_\star$) occurring in the context of μ_{gp}, μ_p, μ. This means that $\mathcal{I}$ approximates the set of prefixes and ambient roles that might occur in each realisable context, $\mathcal{R}$ approximates the bindings of names that might come into effect in each realisable context, and $\mathcal{F}$ the prefix pairs that may react, as P evolves inside $P_\star$.

The judgement is specified in Table 7.6 and refers to Table 7.7 and Table 7.8 for the specification of the closure conditions $\mathsf{closure}_{\lceil M \rceil}$, which ensure that ac-

$$
\begin{array}{lll}
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overset{\twoheadrightarrow}{\mu}} (n)\,P & \text{iff} & (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overset{\twoheadrightarrow}{\mu}} P \\
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu_{gp},\vec{\mu}} [\,P\,]^{\mu_c} & \text{iff} & \mu_c \in \mathcal{I}(\mu_{gp},\vec{\mu}) \wedge (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\vec{\mu},\mu_c} P \\
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overset{\twoheadrightarrow}{\mu}} P \,\big|\, P' & \text{iff} & (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overset{\twoheadrightarrow}{\mu}} P \wedge (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overset{\twoheadrightarrow}{\mu}} P' \\
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overset{\twoheadrightarrow}{\mu}} \sum_{i\in I} M_i^{\ell_i}\,.\,P_i & \text{iff} & \forall i \in I : (\lfloor M_i^{\ell_i} \rfloor \in \mathcal{I}(\overset{\twoheadrightarrow}{\mu}) \wedge \\
 & & \quad (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overset{\twoheadrightarrow}{\mu}} P_i \wedge \mathsf{closure}_{\lceil M_i \rceil}) \\
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overset{\twoheadrightarrow}{\mu}} \mathsf{rec}\,X.\,P & \text{iff} & (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overset{\twoheadrightarrow}{\mu}} P \\
(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overset{\twoheadrightarrow}{\mu}} X & \text{iff} & \mathsf{true}
\end{array}
$$

Table 7.6: 2CFA acceptability judgement, $(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overset{\twoheadrightarrow}{\mu}} P$.

ceptable $(\mathcal{I},\mathcal{R},\mathcal{F})$ correctly capture the dynamics of capabilities. Again, the specification is syntax directed, making the 2CFA *compositional* in the terminology of Flow Logic. The resulting recursive definition is intentionally similar to that of the 0CFA.

The individual clauses of the specification carry the following meaning:

Name restriction: A 2CFA estimate $(\mathcal{I},\mathcal{R},\mathcal{F})$ is acceptable for a process $(n)\,P$ occurring in the context of $\overset{\twoheadrightarrow}{\mu}$ if, and only if, acceptable for the sub-process P in the context of $\overset{\twoheadrightarrow}{\mu}$. Further requirements are avoided to ensure that the analysis is invariant under the heating relation. The treatment of constants is deferred to the closure conditions $\mathsf{closure}_{\lceil M \rceil}$ (where $\lceil M \rceil$ is as M but with all names replaced by '$\cdot$') defined in Section 7.4.3

Ambient boundary: An analysis estimate $(\mathcal{I},\mathcal{R},\mathcal{F})$ is acceptable for a process $[\,P\,]^{\mu_c}$, occurring in the context of $\mu_{gp}, \vec{\mu}$, if, and only if, $\mathcal{I}$ appropriately records the location of μ_c, i.e., $(\mu_{gp}, \vec{\mu}, \mu_c) \in \mathcal{I}$ and $(\mathcal{I},\mathcal{R},\mathcal{F})$ is acceptable for P in the context of $\vec{\mu}, \mu_c$.

Parallel composition: An estimate $(\mathcal{I},\mathcal{R},\mathcal{F})$ is acceptable for $P \,\big|\, Q$ occurring in the context of $\overset{\twoheadrightarrow}{\mu}$ if, and only if, acceptable for each of P and Q in the context of $\overset{\twoheadrightarrow}{\mu}$. It is left to the closure conditions, $\mathsf{closure}_{\lceil M \rceil}$, to ensure a differentiated treatment of parallel composition and summation.

Summation: An estimate $(\mathcal{I},\mathcal{R},\mathcal{F})$ is acceptable for $\sum_{i\in I} M_i^{\ell_i}\,.\,P_i$ occurring in the context of $\overset{\twoheadrightarrow}{\mu}$ if, and only, if for every $i \in I$ it is the case that

- $\mathcal{I}$ records the location of the corresponding prefix, i.e., $(\overset{\twoheadrightarrow}{\mu}, M_i^{\ell_i}) \in \mathcal{I}$,
- $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ is closed under the associated closure condition, $\mathsf{closure}_{\lceil M_i \rceil}$, and
- $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ is acceptable for the sub-process P_i in the context of $\overset{\twoheadrightarrow}{\mu}$.

Recursive process: An analysis estimate $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ is acceptable for $\mathsf{rec}\, \mathrm{X}.\, P$ occurring in the context of $\overset{\twoheadrightarrow}{\mu}$ if, and only if, acceptable for P in the context of $\overset{\twoheadrightarrow}{\mu}$.

Process identifier: any $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ is an acceptable analysis estimate for X in the context of $\overset{\twoheadrightarrow}{\mu}$.

Again, this treatment is correct because the analysis is *flow-insensitive* and the well-formedness condition prevents free process identifiers from occurring within ambients.

7.4.3 Closure Conditions

As for the 0CFA a set of closure conditions govern acceptability of analysis estimates with respect to the dynamic behaviour of processes. Generally these closure conditions follow the schema used for the 0CFA closures. However, the specifications are somewhat more complex and in the following they are, therefore, explained in detail.

As was the case for the 0CFA we introduce a new relation, $\langle\mathcal{R}\rangle$, in order to provide the necessary information about constant definitions. Based on a binding environment $\mathcal{R}$ this *completion of* $\mathcal{R}$

$$\langle\mathcal{R}\rangle \subseteq \mathbf{Role^3} \times \mathbf{Name} \times \mathbf{C}$$

is defined by the closure condition

$$\begin{aligned} \mathsf{complete}\mathcal{R} = {} & (\forall \overset{\twoheadrightarrow}{\mu} : \mathcal{R}(\overset{\twoheadrightarrow}{\mu}) \subseteq \langle\mathcal{R}\rangle(\overset{\twoheadrightarrow}{\mu})) \wedge \\ & \qquad (\forall \overset{\twoheadrightarrow}{\mu}, \nu, n : (\overset{\twoheadrightarrow}{\mu}, \nu) \in \mathcal{I} \wedge n \in \mathbf{C} \Rightarrow \langle\mathcal{R}\rangle(\overset{\twoheadrightarrow}{\mu}, n, n)) \end{aligned} \tag{7.1}$$

where $\nu \in (\mathbf{Role} \cup (\mathbf{Cap} \times \mathbf{Lab}))$. This asserts that $\langle\mathcal{R}\rangle$ contains everything that $\mathcal{R}$ does, and that $\langle\mathcal{R}\rangle$ binds all constants to themselves in every realisable non-empty context.

$$
\begin{array}{l}
\mathsf{closure}_{\mathsf{enter}}\ .\ = \\
\quad \forall \mu_{gp}, \mu_p, \mu, \mu_1, \mu_2, x, y, \ell_1, \ell_2 : \\
\qquad \mathsf{enter}\ x^{\ell_1} \in \mathcal{I}(\mu_p, \mu, \mu_1) \wedge \mu_1 \in \mathcal{I}(\mu_{gp}, \mu_p, \mu) \wedge \\
\qquad \mathsf{accept}\ y^{\ell_2} \in \mathcal{I}(\mu_p, \mu, \mu_2) \wedge \mu_2 \in \mathcal{I}(\mu_{gp}, \mu_p, \mu) \wedge \\
\qquad \langle\mathcal{R}\rangle(\mu_p, \mu, \mu_1, x) \cap \langle\mathcal{R}\rangle(\mu_p, \mu, \mu_2, y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
\qquad \Rightarrow\ \ \mu_1 \in \mathcal{I}(\mu_p, \mu, \mu_2) \wedge \\
\qquad\qquad (\forall \nu : (\ell_1, \nu) \in \mathsf{RPA}_\star \wedge (\mu_p, \mu, \mu_1, \nu) \in \mathcal{I} \Rightarrow (\mu, \mu_2, \mu_1, \nu) \in \mathcal{I}) \wedge \\
\qquad\qquad (\forall \nu, \mu' : (\ell_1, \nu) \in \mathsf{RPA}_\star \wedge (\mu, \mu_1, \mu', \nu) \in \mathcal{I} \Rightarrow (\mu_2, \mu_1, \mu', \nu) \in \mathcal{I}) \wedge \\
\qquad\qquad (\forall p : (\ell_1, p) \in \mathsf{RV}_\star \Rightarrow \mathcal{R}(\mu_p, \mu, \mu_1, p) \subseteq \mathcal{R}(\mu, \mu_2, \mu_1, p)) \wedge \\
\qquad\qquad (\forall p, \mu' : (\ell_1, p) \in \mathsf{RV}_\star \Rightarrow \mathcal{R}(\mu, \mu_1, \mu', p) \subseteq \mathcal{R}(\mu_2, \mu_1, \mu', p)) \wedge \\
\qquad\qquad (\ell_1, \ell_2) \in \mathcal{F} \\
\mathsf{closure}_{\mathsf{accept}}\ .\ = \mathsf{true}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{closure}_{\mathsf{exit}}\ .\ = \\
\quad \forall \mu_{gp}, \mu_p, \mu, \mu_1, \mu_2, x, y, \ell_1, \ell_2 : \\
\qquad \mathsf{exit}\ x^{\ell_1} \in \mathcal{I}(\mu, \mu_2, \mu_1) \wedge \mu_1 \in \mathcal{I}(\mu_p, \mu, \mu_2) \wedge \\
\qquad \mathsf{expel}\ y^{\ell_2} \in \mathcal{I}(\mu_p, \mu, \mu_2) \wedge \mu_2 \in \mathcal{I}(\mu_{gp}, \mu_p, \mu) \wedge \\
\qquad \langle\mathcal{R}\rangle(\mu, \mu_2, \mu_1, x) \cap \langle\mathcal{R}\rangle(\mu_p, \mu, \mu_2, y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
\qquad \Rightarrow\ \ \mu_1 \in \mathcal{I}(\mu_{gp}, \mu_p, \mu) \wedge \\
\qquad\qquad (\forall \nu : (\ell_1, \nu) \in \mathsf{RPA}_\star \wedge (\mu, \mu_2, \mu_1, \nu) \in \mathcal{I} \Rightarrow (\mu_p, \mu, \mu_1, \nu) \in \mathcal{I}) \wedge \\
\qquad\qquad (\forall \nu, \mu' : (\ell_1, \nu) \in \mathsf{RPA}_\star \wedge (\mu_2, \mu_1, \mu', \nu) \in \mathcal{I} \Rightarrow (\mu, \mu_1, \mu', \nu) \in \mathcal{I}) \wedge \\
\qquad\qquad (\forall p : (\ell_1, p) \in \mathsf{RV}_\star \Rightarrow \mathcal{R}(\mu, \mu_2, \mu_1, p) \subseteq \mathcal{R}(\mu_p, \mu, \mu_1, p)) \wedge \\
\qquad\qquad (\forall p, \mu' : (\ell_1, p) \in \mathsf{RV}_\star \Rightarrow \mathcal{R}(\mu_2, \mu_1, \mu', p) \subseteq \mathcal{R}(\mu, \mu_1, \mu', p)) \wedge \\
\qquad\qquad (\ell_1, \ell_2) \in \mathcal{F} \\
\mathsf{closure}_{\mathsf{expel}}\ .\ = \mathsf{true}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{closure}_{\mathsf{merge-}}\ .\ = \\
\quad \forall \mu_{gp}, \mu_p, \mu, \mu_1, \mu_2, x, y, \ell_1, \ell_2 : \\
\qquad \mathsf{merge-}\ x^{\ell_1} \in \mathcal{I}(\mu_p, \mu, \mu_1) \wedge \mu_1 \in \mathcal{I}(\mu_{gp}, \mu_p, \mu) \wedge \\
\qquad \mathsf{merge+}\ y^{\ell_2} \in \mathcal{I}(\mu_p, \mu, \mu_2) \wedge \mu_2 \in \mathcal{I}(\mu_{gp}, \mu_p, \mu) \wedge \\
\qquad \langle\mathcal{R}\rangle(\mu_p, \mu, \mu_1, x) \cap \langle\mathcal{R}\rangle(\mu_p, \mu, \mu_2, y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
\qquad \Rightarrow\ \ (\forall \nu : (\ell_1, \nu) \in \mathsf{RPA}_\star \wedge (\mu_p, \mu, \mu_1, \nu) \in \mathcal{I} \Rightarrow (\mu_p, \mu, \mu_2, \nu) \in \mathcal{I}) \wedge \\
\qquad\qquad (\forall \nu, \mu' : (\ell_1, \nu) \in \mathsf{RPA}_\star \wedge (\mu, \mu_1, mu', \nu) \in \mathcal{I} \Rightarrow (\mu, \mu_2, \mu', \nu) \in \mathcal{I}) \wedge \\
\qquad\qquad (\forall \nu, \mu', \mu'' : (\ell_1, \nu) \in \mathsf{RPA}_\star \wedge (\mu_1, \mu', \mu'', \nu) \in \mathcal{I} \Rightarrow (\mu_2, \mu', \mu'', \nu) \in \mathcal{I}) \wedge \\
\qquad\qquad (\forall p : (\ell_1, p) \in \mathsf{RV}_\star \Rightarrow \mathcal{R}(\mu_p, \mu, \mu_1, p) \subseteq \mathcal{R}(\mu_p, \mu, \mu_2, p)) \wedge \\
\qquad\qquad (\forall p, \mu' : (\ell_1, p) \in \mathsf{RV}_\star \Rightarrow \mathcal{R}(\mu, \mu_1, \mu', p) \subseteq \mathcal{R}(\mu, \mu_2, \mu', p)) \wedge \\
\qquad\qquad (\forall p, \mu', \mu'' : (\ell_1, p) \in \mathsf{RV}_\star \Rightarrow \mathcal{R}(\mu_1, \mu', \mu'', p) \subseteq \mathcal{R}(\mu_2, \mu', \mu'', p)) \wedge \\
\qquad\qquad (\ell_1, \ell_2) \in \mathcal{F} \\
\mathsf{closure}_{\mathsf{merge+}}\ .\ = \mathsf{true}
\end{array}
$$

Table 7.7: 2CFA closure conditions for movement.

Movement Closures. The *2CFA movement closures*, specified in Table 7.7, ensure that acceptable analysis estimates $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ are closed such as to safely over-approximate the consequence of all movement reactions that might occur at run-time. If, e.g., a 2CFA estimate $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ indicates that an enter movement might become enabled in some context $\vec{\vec{\mu}}$, i.e.,

- An enter capability and an accept capability might occur in sibling contexts:
$$\begin{array}{l}\mathsf{enter}\ x^{\ell_1} \in \mathcal{I}(\mu_p, \mu, \mu_1) \wedge \mu_1 \in \mathcal{I}(\mu_{gp}, \mu_p, \mu) \wedge \\ \mathsf{accept}\ y^{\ell_2} \in \mathcal{I}(\mu_p, \mu, \mu_2) \wedge \mu_2 \in \mathcal{I}(\mu_{gp}, \mu_p, \mu)\end{array}$$
- The corresponding prefixes might be concurrently possible:
$$(\ell_1, \ell_2) \in \mathsf{CP}_\star$$
- The enter and accept actions might agree on the name of the communication channel:
$$\langle\mathcal{R}\rangle(\mu_p, \mu, \mu_1, x) \cap \langle\mathcal{R}\rangle(\mu_p, \mu, \mu_2, y) \neq \emptyset$$
As mentioned by Remark 6.10 this requires non-empty intersection in the $\langle\mathcal{R}\rangle$ relation.

Then $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ should reflect that:

- The moving ambient (role) μ_1 might occur in the context of μ_p, μ, μ_2:
$$(\mu_1 \in \mathcal{I}(\mu_p, \mu, \mu_2))$$
- The prefixes and ambient roles that are relevant to ℓ_1 and might occur in the originating context, μ_p, μ, μ_1, might also occur in the new context, μ, μ_2, μ_1:
$$\begin{array}{l}(\forall \nu : (\ell_1, \nu) \in \mathsf{RPA}_\star \wedge (\mu_p, \mu, \mu_1, \nu) \in \mathcal{I} \Rightarrow (\mu, \mu_2, \mu_1, \nu) \in \mathcal{I}) \wedge \\ (\forall \nu, \mu' : (\ell_1, \nu) \in \mathsf{RPA}_\star \wedge (\mu, \mu_1, \mu', \nu) \in \mathcal{I} \Rightarrow (\mu_2, \mu_1, \mu', \nu) \in \mathcal{I})\end{array}$$
Note that we have two contributions depending on whether we consider the children or the grandchildren of μ_1.
- The variables relevant to ℓ_1 that might become bound in the originating context, μ_p, μ, μ_1, might also become bound in the new context, μ, μ_2, μ_1. A similar update is needed for the relevant variables in a child μ' of μ_1:
$$\begin{array}{l}(\forall p : (\ell_1, p) \in \mathsf{RV}_\star \Rightarrow \mathcal{R}(\mu_p, \mu, \mu_1, p) \subseteq \mathcal{R}(\mu, \mu_2, \mu_1, p)) \wedge \\ (\forall p, \mu' : (\ell_1, p) \in \mathsf{RV}_\star \Rightarrow \mathcal{R}(\mu, \mu_1, \mu', p) \subseteq \mathcal{R}(\mu_2, \mu_1, \mu', p))\end{array}$$
Again, note that we have two contributions depending on whether we consider the children or the grandchildren of μ_1.

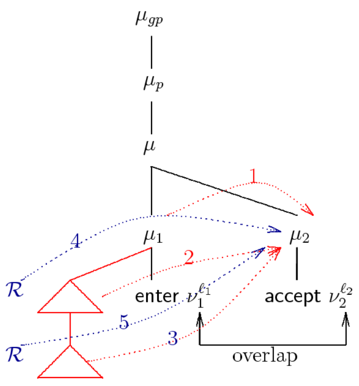


Figure 7.2: 2CFA closure condition for enter movement.

- The corresponding capabilities, ℓ_1 and ℓ_2, might react:
$$(\ell_1, \ell_2) \in \mathcal{F}$$

As only the ambient hosting the enter capability moves, the corresponding accept capability does not cause similar updates.

Communication Closures. Similarly, the *2CFA communication closures*, specified in Table 7.8, ensure that acceptable analysis estimates $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ are closed such as to safely over-approximate the consequence of all communication reactions that might occur at run-time. If, e.g., a 2CFA estimate $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ indicates that an local communication might become enabled in some context $\overrightarrow{\mu}$, i.e.,

- A local input action and a local output action might occur in the same context:
$$x!\{z\}^{\ell_1} \in \mathcal{I}(\overrightarrow{\mu}) \land y?\{p\}^{\ell_2} \in \mathcal{I}(\overrightarrow{\mu})$$
- The corresponding prefixes might be *concurrently possible* in the sense defined by the precomputed filter $\mathsf{CP}_\star$ of Section 6.1:
$$(\ell_1, \ell_2) \in \mathsf{CP}_\star$$
- The input and output actions might agree on the name of the communication channel:
$$\langle\mathcal{R}\rangle(\overrightarrow{\mu}, x) \cap \langle\mathcal{R}\rangle(\overrightarrow{\mu}, y) \neq \emptyset$$
Note here that Remark 6.10 about intersection testing in $\langle\mathcal{R}\rangle$ still applies.

$$\begin{array}{l}
\mathsf{closure}_{\cdot!\{\cdot\}} \triangleq \forall \overset{\twoheadrightarrow}{\mu}, x, y, z, p, \ell_1, \ell_2 : \\
\qquad x!\{z\}^{\ell_1} \in \mathcal{I}(\overset{\twoheadrightarrow}{\mu}) \wedge \\
\qquad y?\{p\}^{\ell_2} \in \mathcal{I}(\overset{\twoheadrightarrow}{\mu}) \wedge \\
\qquad \langle\mathcal{R}\rangle(\overset{\twoheadrightarrow}{\mu}, x) \cap \langle\mathcal{R}\rangle(\overset{\twoheadrightarrow}{\mu}, y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
\qquad \Rightarrow \;\; \langle\mathcal{R}\rangle(\overset{\twoheadrightarrow}{\mu}, z) \subseteq \mathcal{R}(\overset{\twoheadrightarrow}{\mu}, p) \wedge \\
\qquad\qquad (\ell_1, \ell_2) \in \mathcal{F} \wedge \\
\qquad\qquad \mathsf{propagate}\mathcal{R} \\
\mathsf{closure}_{\cdot?\{\cdot\}} \triangleq \mathsf{true}
\end{array}$$

$$\begin{array}{l}
\mathsf{closure}_{\cdot\_!\{\cdot\}} \triangleq \forall \mu_{gp}, \vec{\mu}, \mu_1, x, y, z, p, \ell_1, \ell_2 : \\
\qquad x\_!\{z\}^{\ell_1} \in \mathcal{I}(\mu_{gp}, \vec{\mu}) \wedge \\
\qquad y\hat{}?\{p\}^{\ell_2} \in \mathcal{I}(\vec{\mu}, \mu_1) \wedge \mu_1 \in \mathcal{I}(\mu_{gp}, \vec{\mu}) \wedge \\
\qquad \langle\mathcal{R}\rangle(\mu_{gp}, \vec{\mu}, x) \cap \langle\mathcal{R}\rangle(\vec{\mu}, \mu_1, y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
\qquad \Rightarrow \;\; \langle\mathcal{R}\rangle(\mu_{gp}, \vec{\mu}, z) \subseteq \mathcal{R}(\vec{\mu}, \mu_1, p) \wedge \\
\qquad\qquad (\ell_1, \ell_2) \in \mathcal{F} \wedge \\
\qquad\qquad \mathsf{propagate}\mathcal{R} \\
\mathsf{closure}_{\cdot\hat{}?\{\cdot\}} \triangleq \mathsf{true}
\end{array}$$

$$\begin{array}{l}
\mathsf{closure}_{\cdot\hat{}!\{\cdot\}} \triangleq \forall \mu_{gp}, \vec{\mu}, \mu_1, x, y, z, p, \ell_1, \ell_2 : \\
\qquad x\hat{}!\{z\}^{\ell_1} \in \mathcal{I}(\vec{\mu}, \mu_1) \wedge \mu_1 \in \mathcal{I}(\mu_{gp}, \vec{\mu}) \wedge \\
\qquad y\_?\{p\}^{\ell_2} \in \mathcal{I}(\mu_{gp}, \vec{\mu}) \wedge \\
\qquad \langle\mathcal{R}\rangle(\vec{\mu}, \mu_1, x) \cap \langle\mathcal{R}\rangle(\mu_{gp}, \vec{\mu}, y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
\qquad \Rightarrow \;\; \langle\mathcal{R}\rangle(\vec{\mu}, \mu_1, z) \subseteq \mathcal{R}(\mu_{gp}, \vec{\mu}, p) \wedge \\
\qquad\qquad (\ell_1, \ell_2) \in \mathcal{F} \wedge \\
\qquad\qquad \mathsf{propagate}\mathcal{R} \\
\mathsf{closure}_{\cdot\_?\{\cdot\}} \triangleq \mathsf{true}
\end{array}$$

$$\begin{array}{l}
\mathsf{closure}_{\cdot\#!\{\cdot\}} \triangleq \forall \mu_{gp}, \vec{\mu}, \mu_1, \mu_2, x, y, z, p, \ell_1, \ell_2 : \\
\qquad x\#!\{z\}^{\ell_1} \in \mathcal{I}(\vec{\mu}, \mu_1) \wedge \mu_1 \in \mathcal{I}(\mu_{gp}, \vec{\mu}) \wedge \\
\qquad\quad y\#?\{p\}^{\ell_2} \in \mathcal{I}(\vec{\mu}, \mu_2) \wedge \mu_2 \in \mathcal{I}(\mu_{gp}, \vec{\mu}) \wedge \\
\qquad \langle\mathcal{R}\rangle(\vec{\mu}, \mu_1, x) \cap \langle\mathcal{R}\rangle(\vec{\mu}, \mu_2, y) \neq \emptyset \wedge (\ell_1, \ell_2) \in \mathsf{CP}_\star \\
\qquad \Rightarrow \;\; \langle\mathcal{R}\rangle(\vec{\mu}, \mu_1, z) \subseteq \mathcal{R}(\vec{\mu}, \mu_2, p) \wedge \\
\qquad\qquad (\ell_1, \ell_2) \in \mathcal{F} \wedge \\
\qquad\qquad \mathsf{propagate}\mathcal{R} \\
\mathsf{closure}_{\cdot\#?\{\cdot\}} \triangleq \mathsf{true}
\end{array}$$

Table 7.8: 2CFA closure conditions for communication.

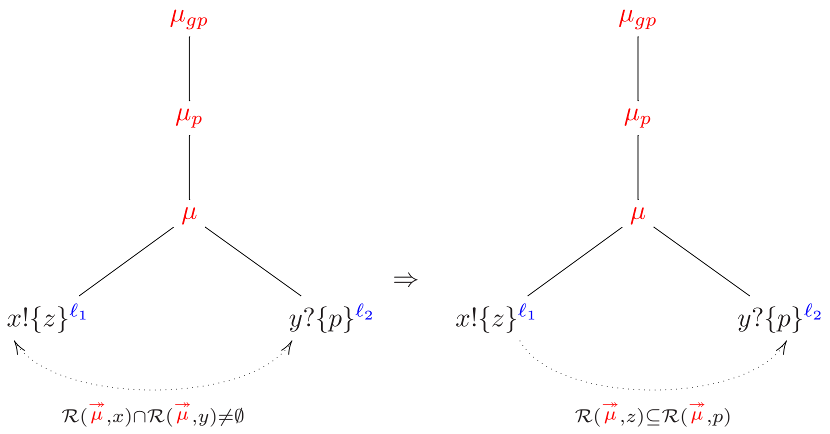

μ_{gp}
μ_p
μ
$x!\{z\}^{\ell_1}$
$y?\{p\}^{\ell_2}$
$\mathcal{R}(\overset{\twoheadrightarrow}{\mu},x)\cap\mathcal{R}(\overset{\twoheadrightarrow}{\mu},y)\neq\emptyset$
$\Rightarrow$
μ_{gp}
μ_p
μ
$x!\{z\}^{\ell_1}$
$y?\{p\}^{\ell_2}$
$\mathcal{R}(\overset{\twoheadrightarrow}{\mu},z)\subseteq\mathcal{R}(\overset{\twoheadrightarrow}{\mu},p)$

Figure 7.3: 2CFA closure condition for local communication.

Then $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ should reflect that:

- Any name that might be bound to z might be the object of the communication and might thus be bound to p:

$$\langle\mathcal{R}\rangle(\overset{\twoheadrightarrow}{\mu}, z) \subseteq \mathcal{R}(\overset{\twoheadrightarrow}{\mu}, p)$$

  Note here that Remark 6.11 about propagation of bindings still applies.

- Any binding of p that might be made is valid, not only in the present context of $\overset{\twoheadrightarrow}{\mu}$, but also in any sub-context that is part of the static scope of p:

$$\mathsf{propagate}\mathcal{R}$$

Remark 7.22 This closed formula is actually a closure condition in itself; it is defined in Table 7.9 and explained in the following subsection. Invoking it here, as a consequence of action closures, simply ensures that it has to be considered only if at least one communication might take place.

Scope Propagation Closure. The scope propagation closure is structurally similar the other closures, but it is simpler: For every context $\mu_{gp}, \vec{\mu}$ and every ambient role μ_c that, according to $(\mathcal{I}, \mathcal{R}, \mathcal{F})$, might occur in it

$$\mu_c \in \mathcal{I}(\mu_{gp}, \vec{\mu})$$

and, according to $\mathsf{SCP}_\star$, might also lie within the shape of the static scope of

$$\mathsf{propagate}\mathcal{R} = \forall \mu_{gp}, \overrightarrow{\mu}, \mu_c, p : \mu_c \in \mathcal{I}(\mu_{gp}, \overrightarrow{\mu}) \wedge \mu_c \in \mathsf{SCP}_\star(p) \Rightarrow \mathcal{R}(\mu_{gp}, \overrightarrow{\mu}, p) \subseteq \mathcal{R}(\overrightarrow{\mu}, \mu_c, p) \quad (7.2)$$

Table 7.9: 2CFA closure condition for propagation of variable bindings.

some variable p

$$\mu_c \in \mathsf{SCP}_\star(p),$$

the estimate $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ is only acceptable if: every binding of p that might occur in the context of $\mu_{gp}, \overrightarrow{\mu}$ might also occur in the context of $\overrightarrow{\mu}, \mu_c$:

$$\mathcal{R}(\mu_{gp}, \overrightarrow{\mu}, p) \subseteq \mathcal{R}(\overrightarrow{\mu}, \mu_c, p)$$

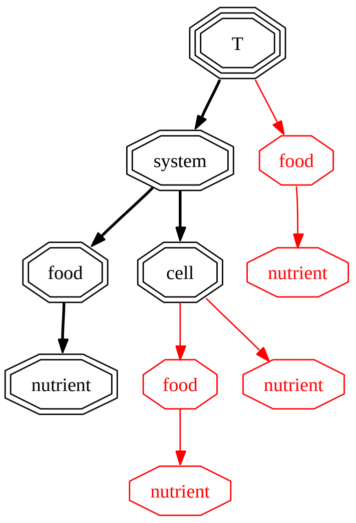
As scoping is static this suffices for correctly propagating the localised environment from 'outside μ_c' into 'inside μ_c' and, thus, ensures that all variable bindings are recorded *throughout static scope*.

Example 7.23 The least fixed point analysis of the running example P_{eat} gives rise to the $\mathcal{I}$ and $\mathcal{R}$ components shown in the tables below.

μ_{gp}	μ_p	μ	$\mathcal{I}(\overrightarrow{\mu}, \mu)$
$\top$	$\top$	$\top$	$food, system$
$\top$	$\top$	$system$	$cell, food$, expel rj^{ℓ_1}
$\top$	$\top$	$food$	enter ac^{ℓ_5}, exit rj^{ℓ_4}, $rea\hat{}?\{rl\}^{\ell_2}$, expel rl^{ℓ_3}, $nutrient$
$\top$	$system$	$food$	$nutrient$, expel rl^{ℓ_3}, $rea\hat{}?\{rl\}^{\ell_2}$, exit rj^{ℓ_4}, enter ac^{ℓ_5}
$\top$	$system$	$cell$	$nutrient, food, rea\_!\{RL\}^{\ell_7}$, expel rj^{ℓ_8}, accept ac^{ℓ_9}
$\top$	$food$	$nutrient$	exit RL^{ℓ_6}
$system$	$food$	$nutrient$	exit RL^{ℓ_6}
$system$	$cell$	$food$	enter ac^{ℓ_5}, exit rj^{ℓ_4}, $rea\hat{}?\{rl\}^{\ell_2}$, expel rl^{ℓ_3}, $nutrient$
$cell$	$food$	$nutrient$	exit RL^{ℓ_6}

μ_{gp}	μ_p	μ	p	$\mathcal{R}(\overrightarrow{\mu}, \mu, p)$
$system$	$cell$	$food$	rl	RL

ℓ	$\mathcal{F}(\ell)$
5	9
6	3
4	1
4	8
7	2



Furthermore, the tree graph gives a graphical relation of the ambient part of the containment relation $\mathcal{I}$. The triple bordered node represents the super-environment, the double bordered nodes connected by bold lines represent the initial configuration (Table 7.6), and the remaining nodes represent the system dynamics (Tables 7.7, 7.8, and 7.9). The trees of the individual frames of Example 3.28 are all sub-trees of this figure. Note, that although the analysis is formally an over-approximation the result is rather precise. □

7.4.4 Properties of the 2CFA

Having *defined* a judgement that specifies the set of acceptable 2CFA estimate $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ we shall now show that it does indeed specify a static analysis that is 1) *well-defined*, 2) *sound with respect to the semantics*, and 3) *implementable*. The required results are very similar to those established for the 0CFA and, hence, the following will be brief.

Well-definedness. As the specification is syntax directed the acceptability judgement is well-defined

Theorem 7.24 (Well-defined) The analysis judgement $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\vec{\vec{\mu}}} P$ is well-defined.

Proof The proof proceeds by structural induction on P. □

Semantic Correctness. The analysis is *correct* as indicated by the following:

Fact 7.25 Assume $Q = \mathsf{rec}\,\mathrm{X}.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

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

Proof Put $\vec{\vec{\mu}}$ for μ in the proof of Fact 6.14. □

Lemma 7.26 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then the following holds:

If $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\vec{\vec{\mu}}} P$ and $P \Rrightarrow Q$ then $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\vec{\vec{\mu}}} Q$.

Proof The proof is by induction on the inference of $P \Rrightarrow Q$ and is similar to that of Lemma 6.15. In the cases of H-ALPH the result follows by referential transparency because renaming is disciplined. The remaining axioms are straightforward. In the case of H-UREC the result follows from Fact 7.25. The induction hypothesis and arduous calculation does the rest. □

Again, to show the invariance under reaction we introduce an expansion of $\mathcal{I}$ into $\mathcal{I}@\mathcal{R}$, which takes into account the bindings of variables specified by the $\mathcal{R}$ component:

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

which satisfies

Fact 7.27 If $\lfloor n \rfloor \in \mathcal{R}(x)$ and $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\vec{\vec{\mu}}} P$ then $(\mathcal{I}@\mathcal{R}, \mathcal{R}, \mathcal{F}) \models^{\vec{\vec{\mu}}} P[^{n}/_{x}]$.

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 7.28 Assume $\mathbf{C} \vdash_{\Gamma_{\mathsf{fn}}} P$, if furthermore $\mathsf{CP}_{\star} \supseteq \mathsf{CP}(P)$. $\mathsf{SCP}_{\star} \supseteq \mathsf{SCP}(P)$. $\mathsf{RV}_{\star} \supseteq \mathsf{RV}(P)$, and $\mathsf{RPA}_{\star} \supseteq \mathsf{RPA}(P)$ then the following holds:

$$\text{If } (\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\vec{\vec{\mu}}} P \text{ and } P \xrightarrow{\tilde{\ell}} Q \text{ then } (\mathcal{I}@\mathcal{R}, \mathcal{R}, \mathcal{F}) \models^{\vec{\vec{\mu}}} Q \text{ and } \tilde{\ell} \in \mathcal{F}$$

.

Proof The proof proceeds by induction of the inference of $P \xrightarrow{\tilde{\ell}} Q$. In the case of movement axioms we compute by equational reasoning. In the case of communication axioms we do the same and use Fact 7.27. In the case of R-AUX we use the induction hypothesis in conjunction with Lemma 7.26. The remaining cases follow by the induction hypothesis. □

Corollary 7.29 (Subject reduction) Assume $\mathsf{PRG}_{\mathbf{C}}(P)$; if furthermore $\mathsf{CP}_{\star} \supseteq \mathsf{CP}(P)$. $\mathsf{SCP}_{\star} \supseteq \mathsf{SCP}(P)$. $\mathsf{RV}_{\star} \supseteq \mathsf{RV}(P)$, and $\mathsf{RPA}_{\star} \supseteq \mathsf{RPA}(P)$ then the following holds:

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

□

And, finally, as $\mathcal{I}@\mathcal{R} = (\mathcal{I}@\mathcal{R})@\mathcal{R}$ we can state the overall correctness result as follows:

Theorem 7.30 (Semantic correctness) Assume that $P_{\star}$ is a well-formed initial program, then $(\mathcal{I}, \mathcal{R}, \mathcal{F}) \models^{\top} P_{\star}$ and $P_{\star} \xrightarrow{\tilde{L}}{}^{\star} P$ entails $(\mathcal{I}@\mathcal{R}, \mathcal{R}, \mathcal{F}) \models^{\top} P$ and $\tilde{L} \subseteq \mathcal{F}$.

Proof The theorem follows by induction on the length of the derivation sequence and uses Corollary 7.29, Corollary 5.13, the induction hypothesis, and the above insight to establish the inductive step. □

```
INPUT :    a Flow Logic $\mathcal{FL}$(BioAmbients, FOL) and
           a BioAmbients process $P_\star$.
OUTPUT :   a FOL formula $\varphi$ such that $(\mathcal{I},\mathcal{R},\mathcal{F}) \models \varphi \Leftrightarrow (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\top} P_\star$.
METHOD :   Set $\varphi := (\bigwedge\{\mathsf{CP}_\star(\ell_1,\ell_2) \mid (\ell_1,\ell_2) \in \mathsf{CP}_\star\}) \wedge$
                   $(\bigwedge\{\mathsf{SCP}_\star(p,\mu) \mid (p,\mu) \in \mathsf{SCP}_\star\}) \wedge$
                   $(\bigwedge\{\mathsf{RV}_\star(\ell,p) \mid (\ell,p) \in \mathsf{RV}_\star\}) \wedge$
                   $(\bigwedge\{\mathsf{RPA}_\star(\ell,\nu) \mid (\ell,\nu) \in \mathsf{RPA}_\star\}) \wedge$
                   $(\bigwedge\{\mathbf{C}_\star(n) \mid n \in \mathbf{C}_\star\}) \wedge$
                   $\mathsf{complete}\mathcal{R} \wedge$
                   $(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\top} P_\star$
           while $\varphi$ contains $(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overrightarrow{\mu}} P'$
             and there is a rule $\alpha$ iff $\beta$ in $\mathcal{FL}$
                   and a substitution $\theta$
             such that $\theta\alpha = (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overrightarrow{\mu}} P'$
            do replace $(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overrightarrow{\mu}} P'$ with $\theta\beta$ in $\varphi$.
```

Table 7.10: Generation of 2CFA constraints.

Implementability. Obviously the set of acceptable analysis estimates constitutes a *Moore Family*:

Theorem 7.31 (Moore family (2CFA)) For any program, $P_\star$, the set of acceptable analyses under $\models^{\overrightarrow{\mu}}$ is a Moore family, i.e.,

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

Proof The proof proceeds by structural induction in P and is similar to that for the 0CFA. □

Recall, from Chapter 4, that a Moore family is never empty and is also guaranteed to have a least element $(\mathcal{I}_\star,\mathcal{R}_\star,\mathcal{F}_\star) = \bigsqcap\{(\mathcal{I},\mathcal{R},\mathcal{F}) \mid (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\overrightarrow{\mu}} P_\star\}$, which is the most informative acceptable estimate. The computation of $(\mathcal{I}_\star,\mathcal{R}_\star,\mathcal{F}_\star)$ is made possible by a clause generation algorithm as outlined in Table 7.10. In order to compute $(\mathcal{I}_\star,\mathcal{R}_\star,\mathcal{F}_\star)$ for a given $P_\star$ we use the constraint generator to produce a suitable φ and then compute the least model using the Succinct Solver Suite [NNS$^+$04]. Again, this implementation strategy guarantees polynomial complexity, but the degree is higher than for the 0CFA.

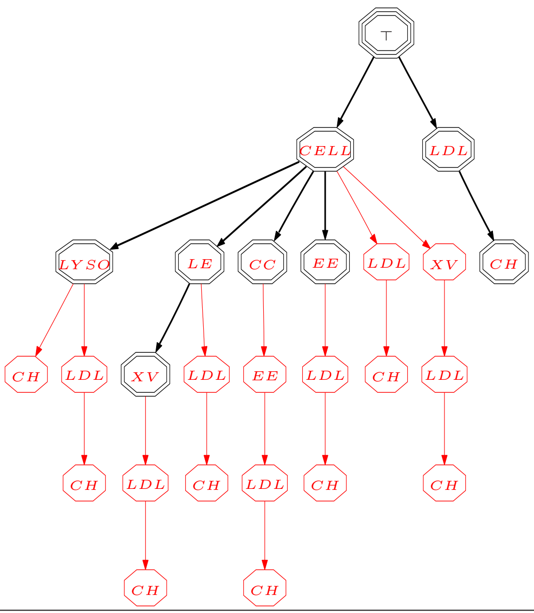


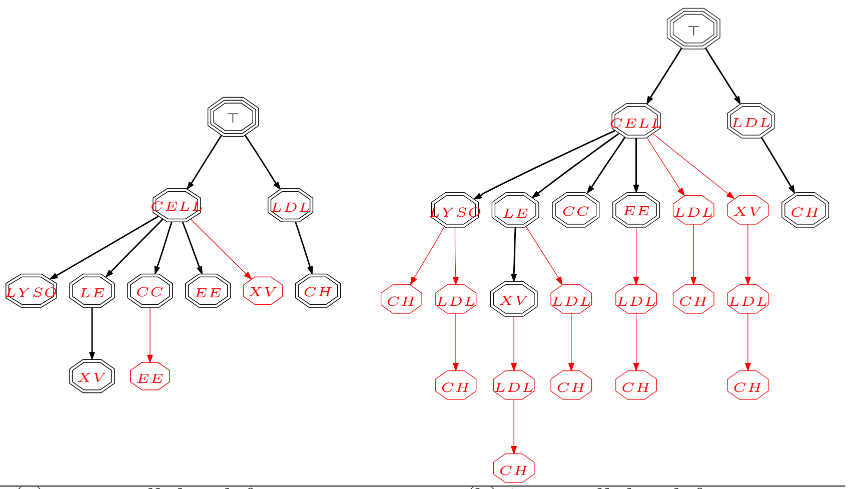
Figure 7.4: 2CFA analysis of normal receptor LDL pathway model. The special $\top$ node with triple borders represents the super-environment, The other nodes represent ambient roles and the edges represent the containment relation $\mathcal{I}$. The nodes with double borders connected with bold (black) edges represent the system in its initial state. The remaining (red) nodes and edges account for the dynamic evolution of the system.

7.5 CASE: Analysing the LDL Degradation Pathway

We now continue our investigation of the LDL pathway model by subjecting it to the 2CFA. The resulting $(\mathcal{I}, \mathcal{R}, \mathcal{F})$ is only included in Appendix B.2.1, but the $\mathcal{I}$ relation is depicted graphically in Fig. 7.4.

At first look the analysis result appears remarkably more precise than the corresponding 0CFA result of the previous chapter. Formally, this is an over-approximation - but the picture depicts exactly the behaviour that we expect the normal receptor LDL model to exhibit. In particular, we notice that cholesterol (CH) might be released only in the lumen of the lysosome ($LYSO$), which is essential.

Note that the graph indicates that LDL might float freely in the cytosol (the



(a) extra-cellular defects (b) intra-cellular defects

Figure 7.5: 2CFA analysis of defect receptor LDL pathway models.

top-level fluid of the $CELL$). This is the modelling artifact noted in Section 3.3 appearing again. Also note that the LDL *can* occur inside an un-coated early endosome (EE) after the clathrin coat (CC) has been shredded; in this case the double bordered EE node represents both the initial configuration and a later stage of evolution.

7.5.1 Related Diseases

Fig. 7.5(a) shows the effect of disabling the exo-plasmic binding site of the LDL receptor. This corresponds to the model of Appendix A.2 and the analysis result is presented verbatim in Appendix B.2.2. As the graph reveals, the cell can no longer internalise LDL particles. The internal part of the pathway still works; hence the cell still internalises early endosomes, but only empty ones (EE may occur inside CC, but it carries no LDL cargo). In fact, the pathway processes the endosomes right up to the point where transfer vesicles merge with the lysosome (XV might occur inside $CELL$). Intuitively, this is exactly the expected result.

Similarly, Fig. 7.5(b) shows the effect of disabling the cytosolic binding site of the LDL receptor. This corresponds to the model of Appendix A.3 and the analysis result is presented verbatim in Appendix B.2.3. We expect the receptor protein to be able to bind LDL particles, but internalisation of early endosomes should fail. In contrast to before, however, the analysis fails to capture this. It appears that the clathrin coating process is simply skipped and the EE is internalised directly. This is similar to the failure we observed for the 0CFA analysis. Fortunately the 2CFA is much more precise, and we can now see the problem: The modelling artifact allows the LDL into the cytosol, where it can enter the EE. Since the analysis is completely flow insensitive it fails to capture that the EE can only merge with the late endosome LE *after* it has been clad by, and subsequently shredded, the CC.

7.6 Concluding Remarks

The 2CFA presented in this chapter extends the tradition of CFAs that spawned the 0CFA of the previous chapter [NNH02, NNB04, LM01, NNPdR07, NNP04]. In particular, precision has been increased by incorporating context in the manner of 2CFA and by using auxiliary information produced by four separate static analyses:

As for the 0CFA of the previous chapter a safe approximation to the concurrently possible prefixes, $\mathsf{CP}_\star$, is used to impose a restriction on the scope of the closure conditions. We only compute the closure of semantic actions that are not impossible because the involved capabilities are separated by non-deterministic choice. A safe approximation to the spatial shape of static scope, $\mathsf{SCP}_\star$, is used to ensure that name bindings are only propagated into sub-context that might actually be part of the static scope of the corresponding bound variables.

A safe approximation to the relevant variables, $\mathsf{RV}_\star$, is used to restrict the copying of name bindings (in $\mathcal{R}$) that is demanded by the closure conditions for ambient movement. Only the bindings of variables that do not disappear due to non-deterministic choice are copied to the new location.

Finally, a safe approximation to the relevant prefixes and ambients, $\mathsf{RPA}_\star$. is used to achieve a similar restriction on the copying of prefixes and ambients (in $\mathcal{I}$) that is demanded by the closure conditions for movement.

These relations all emerge as fix-point properties; hence the information is computed by a hierarchy of indexed functions

$$\begin{tikzcd}
\mathsf{SCP}_\star & \mathsf{RPA}_\star & \mathsf{RV}_\star & \mathsf{CP}_\star \\
\mathsf{SCP}_{\Gamma_{\mathsf{amb}},\Delta_{\mathsf{SCP}}}() \arrow[u] & \mathsf{RPA}_{\Gamma_{\mathsf{cap}},\Gamma_{\mathsf{amb}},\Delta_{\mathsf{RPA}}}() \arrow[u] & \mathsf{RV}_{\Gamma_{\mathsf{fv}},\Gamma_{\mathsf{cap}},\Delta_{\mathsf{RV}}}() \arrow[u] & \mathsf{CP}_{\Gamma_{\mathsf{cap}},\Delta_{\mathsf{CP}}}() \arrow[u] \\
\mathsf{amb}_{\Gamma_{\mathsf{amb}}}() \arrow[u] \arrow[ur] & & \mathsf{cap}_{\Gamma_{\mathsf{cap}}}() \arrow[ul] \arrow[u] \arrow[ur] & \mathsf{fv}_{\Gamma_{\mathsf{fv}}}() \arrow[ul]
\end{tikzcd}$$

Compared to the original presentation in [PNN06b] these functions have been refactored in this dissertation, such that the hierarchy is more natural. Also, the scope approximation is now sound, which, unfortunately, it was not in [PNN06b].

The results obtained by applying the 2CFA to both the running example P_{eat} and the LDL pathway model show a very clear improvement over the 0CFA of the previous chapter; due to the spatial separation of contexts the resulting graphs are both easier to understand and much more precise. Still, however, the result obtained from the LDL pathway with cytosolic receptor defects is disappointing. Here, the 2CFA quite simply falls short because it is flow-insensitive, and there are two possible responses to this problem:

One can take the engineers approach and fashion a model that is easier to analyse, e.g., by incorporating explicit synchronisation points. In [PNN05] this approach was used to good effect for a simpler version of the 2CFA.

Fundamentally, however, one should apply Ockham's Razor and always keep models as pure as possible. This is the approach we advocate by specifying a flow-sensitive analysis in the following chapter.

Part III

Analysing for Causal Properties

Chapter 8

Pathway Analysis

"Any important disease whose causality is murky, and for which treatment is ineffectual, tends to be awash in significance."
— Susan Sontag

This chapter presents a flow sensitive *Pathway Analysis* that given a model, $P_\star$, computes a safe approximation to the set of, possibly infinite, reaction sequences that, according to $P_\star$, may occur. The analysis is evaluated in the context of both the LDL degradation pathway model of Section 3.3 and the transcription model of Section 3.4. Some of the presented material has been submitted as part of [PNN07].

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

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

where we write '.' for an arbitrary name. Here there is one exposed occurrence of each of the prefixes $n!\{m\}^{\ell_1}$, $.?\{.\}^{\ell_3}$, 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, $P \overset{(\ell_1,\ell_4)}{\longrightarrow} Q$, such that the resulting configuration,

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

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. This 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 (Section 4.2), normally associated with Data Flow Analysis, in order to deal with this. In doing so, we shall attach transfer functions to the reactions of processes. Corresponding to a *forward Data Flow Analysis* [NNH99], these transfer functions will compute how the extended multisets of exposed prefixes are transformed as reactions occur. Much akin to Bitvector Frameworks (Section 4.2) our transfer functions will take the simple form

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

However, we shall need to generalise the developments to extended multisets, rather than the simple powersets usually associated with bitvector frameworks.

As opposed to the Flow Logic (Section 4.3) approach of the previous chapters the approach of Monotone Frameworks offers no convenient separation between specification and implementation. Thus, once we have defined the notion of exposed prefixes and the corresponding transfer functions, we shall turn to the issue of implementation and define a suitable worklist algorithm.

Given a program $P_\star$ the idea is to construct a *Finite Automaton* such that the potentially infinite transition system of $P_\star$ is faithfully represented within the states and transitions of the automaton. As outlined in Section 4.2, the computed automaton will take the form

$$(\mathsf{Q}_\star, \mathsf{q}_\star, \delta_\star, \mathsf{E}_\star)$$

where $\mathsf{Q}_\star$ is a set of states, $\mathsf{q}_\star$ the initial state, $\delta_\star$ a transition relation, and $\mathsf{E}_\star$ a vector of extended multisets corresponding to the states. It will emerge from the construction that the resulting automaton is *partially deterministic*.

The chapter will be in six sections. First we introduce the notion of extended multisets in Section 8.1. Then, in Section 8.2, we specify the transfer functions associated with the Pathway Analysis. In 8.3 we define a worklist algorithm for the analysis. Then we turn to the case studies. First we analyse the LDL pathway model in Section 8.4, and then the transcription model in Section 8.5. Finally, in Section 8.6, we summarise our findings.

8.1 Extended Multisets

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

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

which 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:

$\bot_{\mathfrak{M}}$ is defined by:

$$\forall \ell : \bot_{\mathfrak{M}}(\ell) = 0$$

$\top_{\mathfrak{M}}$ is defined by:

$$\forall \ell : \top_{\mathfrak{M}}(\ell) = \infty$$

$\mathsf{min}_{\mathfrak{M}}$ is defined by:

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

$\mathsf{max}_{\mathfrak{M}}$ is defined by:

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

Furthermore we define addition and subtraction on multisets as follows:

$+_{\mathfrak{M}}$ is defined by:

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

$-_{\mathfrak{M}}$ 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 8.2.3 is always a correct over-approximation.

$$\begin{array}{rcl}
\mathcal{E}_{\Gamma_\mathcal{E}}[\![\,(n)\,P\,]\!] & = & \mathcal{E}_{\Gamma_\mathcal{E}}[\![\,P\,]\!] \\
\mathcal{E}_{\Gamma_\mathcal{E}}[\![\,[\,P\,]^{\mu}\,]\!] & = & \mathcal{E}_{\Gamma_\mathcal{E}}[\![\,P\,]\!] \\
\mathcal{E}_{\Gamma_\mathcal{E}}[\![\,P_1 \mid P_2\,]\!] & = & \mathcal{E}_{\Gamma_\mathcal{E}}[\![\,P_1\,]\!] +_{\mathfrak{M}} \mathcal{E}_{\Gamma_\mathcal{E}}[\![\,P_2\,]\!] \\
\mathcal{E}_{\Gamma_\mathcal{E}}[\![\,\sum_{i\in I} M_i^{\ell_i}\,.\,P_i\,]\!] & = & \sum_{\mathfrak{M}\,i\in I} \bot_{\mathfrak{M}}[\ell_i \mapsto 1] \\
\mathcal{E}_{\Gamma_\mathcal{E}}[\![\,\mathsf{rec}\,\mathrm{X}.\,P\,]\!] & = & \mathrm{LFP}(\lambda E.\ \mathcal{E}_{\Gamma_\mathcal{E}[X\mapsto E]}[\![\,P\,]\!]) \\
 & = & \mathcal{E}_{\Gamma_\mathcal{E}[X\mapsto\bot_\mathfrak{M}]}[\![\,P\,]\!] \bowtie_{\mathfrak{M}} \mathcal{E}_{\Gamma_\mathcal{E}[X\mapsto\mathcal{E}_{\Gamma_\mathcal{E}[X\mapsto\bot_\mathfrak{M}]}[\![\,P\,]\!]]}[\![\,P\,]\!] \\
\mathcal{E}_{\Gamma_\mathcal{E}}[\![\,X\,]\!] & = & \Gamma_\mathcal{E}(X) \\
\hline
(\mathsf{M} \bowtie_{\mathfrak{M}} \mathsf{M}')(\ell) & = & \begin{cases} \mathsf{M}(\ell) & \text{if } \mathsf{M}(\ell) = \mathsf{M}'(\ell) \\ \infty & \text{otherwise} \end{cases}
\end{array}$$

Table 8.1: Exposed capabilities, $\mathcal{E}_{\Gamma_\mathcal{E}}[\![\,P\,]\!]$, of a process, P.

8.2 Computing and Preserving Exposed Prefixes

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

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

intended to capture the following intuition: For an identifier closed process, P, the expression $\mathcal{E}_\star[\![P]\!]\,(\ell)$ 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 just a given process P, but on all processes that may arise by heating of P; in particular those that arise by the unfolding of recursion. In order to appropriately address this issue we use the parameterised function $\mathcal{E}_{\Gamma_\mathcal{E}}[\![\ ]\!]$, shown in Table 8.1, when computing the exposed prefixes. The environment $\Gamma_\mathcal{E}\,:\,\mathbf{Pid}\to\mathfrak{M}$ is a mapping that is used to associate each free process identifier with an extended multiset of exposed prefixes. 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 that the naive computation may not terminate because $(\mathfrak{M}, \leq_\mathfrak{M})$ admits infinite ascending chains. However, it turns out that we are justified in recasting the computation in terms of the *expansion operator*, $\bowtie_\mathfrak{M}$, defined in Table 8.1, which ensures termination in two iterations. We refer to [NN08] for a formal proof of this result, but provide an informal argument below.

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 $\mathsf{rec}\,X.\,(X \,\big|\, P)$, *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.

Besides this technicality, the function $\mathcal{E}_{\Gamma_\mathcal{E}}[\![\;]\!]$ performs a rather straightforward recursive descent into processes; it uses the addition operation of extended multisets in order to calculate the total number of exposed prefixes, i.e., those that might participate in the next reaction. As should be evident from the case concerning summations, only the top-most prefixes contribute and the result is obtained by addition of their multiplicities.

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

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

Convention 8.1 In the case of initial programs, $P_\star$, we shall use the distinguished symbol $E_\star$ to denote $\mathcal{E}_\star[\![P_\star]\!]$. ■

Example 8.2 The result E_{eat} of subjecting the program P_{eat} to the exposed capability analysis 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, which is not exposed – is exposed. □

Correctness of $\mathcal{E}_\Gamma[\![\,P\,]\!]$. 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 8.3 Assume $Q = \mathsf{rec}\,X.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

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

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

And then we go on to show the main result:

Lemma 8.4 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then both of the following hold:

1. If $P \Rrightarrow Q$ then $\mathcal{E}_{\Gamma_\mathcal{E}}[\![Q]\!] \leq_{\mathfrak{M}} \mathcal{E}_{\Gamma_\mathcal{E}}[\![P]\!]$
2. If $P \xrightarrow{(\ell_1,\ell_2)} Q$ then $\ell_1 \in \mathsf{dom}(\mathcal{E}_{\Gamma_\mathcal{E}}[\![P]\!])$ and $\ell_2 \in \mathsf{dom}(\mathcal{E}_{\Gamma_\mathcal{E}}[\![P]\!])$

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 5.11, that $LFP(\lambda E.\ \mathcal{E}_{\Gamma_\mathcal{E}[X \mapsto E]}[\![P]\!])$ is indeed a fixed point, and Fact 8.3.

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. □

Corollary 8.5 If $\mathsf{PRG}_\mathbf{C}(P)$ and $P \Rrightarrow Q$ then $\mathcal{E}_{\Gamma_\mathcal{E}}[\![Q]\!] \leq_{\mathfrak{M}} \mathcal{E}_{\Gamma_\mathcal{E}}[\![P]\!]$ and furthermore, if $P \xrightarrow{(\ell_1,\ell_2)} Q$ then $\ell_1 \in \mathsf{dom}(\mathcal{E}_{\Gamma_\mathcal{E}}[\![P]\!])$ and $\ell_2 \in \mathsf{dom}(\mathcal{E}_{\Gamma_\mathcal{E}}[\![P]\!])$. □

8.2.1 Generated Prefixes

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

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

where elements of type

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

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 $\mathcal{G}_\star[\![P]\!]\,(\ell_1)(\ell)$ 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. Whenever $\mathcal{G}_\star[\![P]\!]\,(\ell_1)(\ell) = m$ and $\mathcal{G}_\star[\![P]\!]\,(\ell_2)(\ell) = n$, we write $\mathcal{G}_\star[\![P]\!]\,((\ell_1,\ell_2))(\ell)$ to denote $\mathcal{G}_\star[\![P]\!]\,(\ell_1)(\ell) + \mathcal{G}_\star[\![P]\!]\,(\ell_2)(\ell) = m + n$.

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 $\mathfrak{M}$. For example, $\leq_\mathfrak{T}$ is defined by:

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

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

Table 8.2: Generated capabilities, $\mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\, P \,]\!]$, of a process, P.

Once more, we have to take the unfolding of recursion into account when computing the generate function. For this, we use the parameterised recursive procedure $\mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\;]\!]$ defined in Table 8.2. Here $\Gamma_{\mathcal{E}} : \mathbf{Pid} \to \mathfrak{M}$ is as before, i.e., a mapping that associates an extended multiset with each free process identifier, and $\Delta_{\mathcal{G}} : \mathbf{Pid} \to \mathfrak{T}$ is a mapping that associates a mapping from labels into extended multisets with each free process identifier. We use the $\Gamma_{\mathcal{E}}$ environment to ensure that the number of prefixes exposed in a given continuation is computed correctly, that is, in the case of $\mathsf{rec}\, \mathrm{X}.\, P$, $\Gamma_{\mathcal{E}}$ is used to appropriately associate X with the multiset of prefixes, $\mathcal{E}_{\Gamma_{\mathcal{E}}}[\![\, \mathsf{rec}\, \mathrm{X}.\, P \,]\!]$, exposed by the recursion construct. The $\Delta_{\mathcal{G}}$ environment, on the other hand, is used to support the fixed point computation required for $\mathsf{rec}\, \mathrm{X}.\, P$ — much like the $\Gamma_{\mathcal{E}}$ environment of $\mathcal{E}_{\star}[\![]\!]$.

The interesting case is that of the guarded sum construct, $\sum_{i\in I} M_i^{\ell_i} \,.\, P_i$. 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 $\mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\, P \,]\!]$ is a safe over-approximation, in the sense that, for every ℓ, $\mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\, P \,]\!](\ell)$ 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 arises from doing so. This amounts to the least fixed point computation shown in Table 8.2. Again, the termination of the naive computation is endangered, because $(\mathfrak{T}, \leq_{\mathfrak{T}})$ admits infinite ascending chains. As formally proved in [NN08], however, the computation actually stabilises after a single iteration, which justifies the alternative formulation of Table 8.2.

The desired function $\mathcal{G}_\star[\![\,]\!]$ is defined by:

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

Convention 8.6 In the case of initial programs, $P_\star$, we shall use the distinguished symbol $G_\star$ to denote $\mathcal{G}_\star[\![P_\star]\!]$. ■

Example 8.7 The result, G_{eat}, of subjecting the program P_{eat} to the generated capability analysis is shown below:

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

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

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

Fact 8.8 Assume $Q = \mathsf{rec}\,X.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$\mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\,P[^{Q}/_{X}]\,]\!] = \begin{cases} \mathcal{G}_{\Gamma_{\mathcal{E}}[X \mapsto \mathcal{E}_{\Gamma_{\mathcal{E}}}[\![\,Q\,]\!]],\Delta_{\mathcal{G}}[X \mapsto \bot_{\mathfrak{T}}]}[\![\,P\,]\!] \;\mathsf{max}_{\mathfrak{T}}\; \mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\,Q\,]\!] & \text{if } X \in \mathsf{fpi}(P) \\ \mathcal{G}_{\Gamma_{\mathcal{E}}[X \mapsto \mathcal{E}_{\Gamma_{\mathcal{E}}}[\![\,Q\,]\!]],\Delta_{\mathcal{G}}[X \mapsto \bot_{\mathfrak{T}}]}[\![\,P\,]\!] & \text{otherwise} \end{cases} \tag{8.1}$$

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

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

Lemma 8.9 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then the following holds:

$$\text{If } P \Rrightarrow Q \text{ then } \mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\, Q \,]\!] \leq_{\mathfrak{T}} \mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\, P \,]\!].$$

Proof The proof is by induction on the inference of $P \Rrightarrow Q$. When showing the lemma for H-UREC we use Fact 5.11, Fact 8.8, and that $\mathrm{LFP}(\lambda G.\, \mathcal{G}_{\Gamma_{\mathcal{E}}[X \mapsto \mathcal{E}_{\Gamma_{\mathcal{E}}}[\![\, \mathsf{rec}\, X.\, P \,]\!]],\Delta_{\mathcal{G}}[X \mapsto G]}[\![\, P \,]\!])$ is indeed a fixed point. In the case of H-CSUM we use Lemma 8.4-*(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 8.10 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then the following holds:

$$\text{If } P \xrightarrow{\tilde{\ell}} Q \text{ then } \mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\, Q \,]\!] \leq_{\mathfrak{T}} \mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\, P \,]\!].$$

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 $\mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\, P \,]\!] = \mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\, P[{}^{n}/_{p}] \,]\!]$ and $\mathcal{E}_{\Gamma_{\mathcal{E}}}[\![\, P \,]\!] = \mathcal{E}_{\Gamma_{\mathcal{E}}}[\![\, P[{}^{n}/_{p}] \,]\!]$ in the case of communication.

For the rule using the structural congruence we use Lemma 8.9. The remaining rules follow by the induction hypothesis. □

Corollary 8.11 If $\mathsf{PRG}_{\mathbf{C}}(P)$ and $P \xrightarrow{\tilde{\ell}} Q$ then $\mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\, Q \,]\!] \leq_{\mathfrak{T}} \mathcal{G}_{\Gamma_{\mathcal{E}},\Delta_{\mathcal{G}}}[\![\, P \,]\!]$. □

8.2.2 Killed Prefixes

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

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

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. Whenever $\mathcal{K}_{\star}[\![P]\!]\,(\ell_1)(\ell) = m$ and $\mathcal{K}_{\star}[\![P]\!]\,(\ell_2)(\ell) = n$, we write $\mathcal{K}_{\star}[\![P]\!]\,((\ell_1,\ell_2))(\ell)$ to denote $\mathcal{K}_{\star}[\![P]\!]\,(\ell_1)(\ell) + \mathcal{K}_{\star}[\![P]\!]\,(\ell_2)(\ell) = m + n$.

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

Table 8.3: Killed capabilities, $\mathcal{K}_{\Delta_\mathcal{K}}[\![P]\!]$, of a process, P.

The unfolding of recursion also complicates the computation of kill information, and therefore we use the parameterised function $\mathcal{K}_{\Delta_\mathcal{K}}[\![\ ]\!]$, defined in Table 8.3, for the computation. Here, the environment $\Delta_\mathcal{K} : \mathbf{Pid} \to \mathfrak{T}$ 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 $\mathsf{rec}\, X.\, P$. Note, that we do not need a $\Gamma_\mathcal{E}$ environment, because the definition of $\mathcal{K}_{\Delta_\mathcal{K}}[\![\ ]\!]$ does not rely on $\mathcal{E}_{\Gamma_\mathcal{E}}[\![\ ]\!]$.

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 $\mathsf{min}_{\mathfrak{T}}$. As a consequence, the computed $\mathcal{K}_{\Delta_\mathcal{K}}[\![\ ]\!]$ is always a safe approximation – in the sense that, for every ℓ, $\mathcal{K}_{\Delta_\mathcal{K}}[\![P]\!](\ell)$ 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 arises from doing so. This amounts to the least fixed point computation shown in Table 8.3, which is guaranteed to terminate because $\mathfrak{T}$ admits no infinite descending chains.

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

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

Convention 8.12 In the case of initial programs, $P_\star$, we shall use the distinguished symbol $K_\star$ to denote $\mathcal{K}_\star[\![P_\star]\!]$. ■

Example 8.13 The result, K_{eat}, of subjecting the program P_{eat} to the killed capability analysis is shown below:

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

□

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

Fact 8.14 Assume $Q = \mathsf{rec}\,X.\,Q'$ and $\mathbf{C} \vdash Q$; if furthermore $\mathsf{fpi}(Q) = \emptyset$ and $P \prec Q$ then

$$\mathcal{K}_{\Delta_{\mathcal{K}}}[\![P[{}^{Q}/{}_{X}]]\!] = \begin{cases}\mathcal{K}_{\Delta_{\mathcal{K}}[X\mapsto\bot_{\mathfrak{T}}]}[\![P]\!] \mathbin{\mathsf{min}_{\mathfrak{T}}} \mathcal{K}_{\Delta_{\mathcal{K}}}[\![Q]\!] & \text{if } X \in \mathsf{fpi}(P)\\ \mathcal{K}_{\Delta_{\mathcal{K}}[X\mapsto\bot_{\mathfrak{T}}]}[\![P]\!] & \text{otherwise}\end{cases}$$

Proof The result follows by structural induction on P. □

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

Lemma 8.15 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then the following holds:

$$\text{If } P \Rrightarrow Q \text{ then } \mathcal{K}_{\Delta_{\mathcal{K}}}[\![P]\!] \leq_{\mathfrak{T}} \mathcal{K}_{\Delta_{\mathcal{K}}}[\![Q]\!].$$

Proof The proof is by induction on the inference of $P \Rrightarrow Q$. In the case of H-UREC we use Fact 5.11, that $\mathrm{LFP}(\lambda K.\ \mathcal{K}_{\Delta_{\mathcal{K}}[X\mapsto K]}[\![P]\!])$ is indeed a fixed point, and Fact 8.14. 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 8.16 If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then the following holds:

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

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 $\mathcal{K}_{\Delta_\mathcal{K}}[\![\, P \,]\!] = \mathcal{K}_{\Delta_\mathcal{K}}[\![\, P[^n/_p] \,]\!]$ in the case of communication.

In the case of R-AUX we use Lemma 8.15. The remaining rules follow by the induction hypothesis. □

Corollary 8.17 If $\mathsf{PRG}_{\mathbf{C}}(P)$ and $P \xrightarrow{\tilde{\ell}} Q$ then $\mathcal{K}_{\Delta_\mathcal{K}}[\![\, P \,]\!] \leq_{\mathfrak{T}} \mathcal{K}_{\Delta_\mathcal{K}}[\![\, Q \,]\!]$. □

8.2.3 The Transfer Function

In the classical Bit Vector Frameworks (Section 4.2) transfer functions take the form:

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

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

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

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

Given a multiset of exposed prefixes, E, the transfer function computes, for an a priori given process, P, and transition, $\tilde{\ell}$, a safe over-approximation to the set of prefixes exposed after the transition.

Correctness of the Transfer Function. 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 8.18 (Subject reduction) If $\mathbf{C} \vdash P$ and $\mathsf{fpi}(P) = \emptyset$ then the following holds:

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

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

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

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

$$\begin{aligned}
&\mathcal{E}_\star[\![P]\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![P'']\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![Q]\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![Q'']\!] \\
&\leq_{\mathfrak{M}} (\mathcal{E}_\star[\![\mathsf{enter}\ n^{\ell_1}\,.\,P + P']\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![P'']\!] -_{\mathfrak{M}} \mathcal{E}_\star[\![\mathsf{enter}\ n^{\ell_1}\,.\,P + P']\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![P]\!]\,) \\
&+_{\mathfrak{M}} (\mathcal{E}_\star[\![\mathsf{accept}\ n^{\ell_2}\,.\,Q + Q']\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![Q'']\!] -_{\mathfrak{M}} \mathcal{E}_\star[\![\mathsf{accept}\ n^{\ell_2}\,.\,Q + Q']\!] +_{\mathfrak{M}} \mathcal{E}_\star[\![Q]\!]\,) \\
&\qquad\qquad \leq_{\mathfrak{M}} (\mathcal{E}_\star[\![lhs]\!] -_{\mathfrak{M}} \mathcal{K}_\star[\![lhs]\!]\,(\ell_1,\ell_2)) +_{\mathfrak{M}} \mathcal{G}_\star[\![lhs]\!]\,(\ell_1,\ell_2)
\end{aligned} \tag{8.2}$$

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

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

and

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

so that

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

Thus we may calculate

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

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 8.4, 8.9, and 8.15. □

Corollary 8.19 (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 8.20 (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 8.19. The inductive step is established using Corollaries 8.10, 8.16, 8.19, and 5.13. □

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

8.3 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, $P_\star$, the idea is to construct a finite automaton, such that the potentially infinite transition system of $P_\star$ is faithfully represented within the states and transitions of the automaton. The computed automaton will have the following components:

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

We shall refer to this automaton as $(\mathsf{Q}_\star, \mathsf{q}_\star, \delta_\star, \mathsf{E}_\star)$.

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.

8.3.1 The Worklist Algorithm

Motivated by Monotone Frameworks (Section 4.2) the heart of the computation is an iterative worklist algorithm that computes a least 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 Table 8.4, 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, $\mathsf{q}_\star$, and

```
INPUT : Instance $(\mathsf{q}_\star, E_\star, \mathcal{F}_\star)$
OUTPUT : Least solution $(\mathsf{Q}_\star, \mathsf{q}_\star, \delta_\star, \mathsf{E}_\star)$
         such that $(\mathsf{Q}_\star, \mathsf{q}_\star, \delta_\star, \mathsf{E}_\star) \models \textsc{pathway}(P_\star)$
METHOD : Step 1: Initialisation of $\mathsf{Q}, \mathsf{W}, \delta,$ and $\mathsf{E}$
                 (1)  $\mathsf{Q} := \{\mathsf{q}_\star\};\ \mathsf{E}[\mathsf{q}_\star] := E_\star;$
                 (2)  $\mathsf{W} := \{\mathsf{q}_\star\};\ \delta := \emptyset;$
         Step 2: Iteration (updating $\mathsf{Q}, \mathsf{W}, \delta,$ and $\mathsf{E}$)
                 (3)  while $\mathsf{W} \neq \emptyset$ do
                 (4)    select $q_s$ from $\mathsf{W}$; $\mathsf{W} := \mathsf{W} \backslash \{q_s\};$
                 (5)    for each $\tilde{\ell} \in \mathsf{enabled}(E[q_s])$ do
                 (6)      let $E = \mathsf{transfer}_{P_\star, \tilde{\ell}}(\mathsf{E}[q_s])$
                 (7)      in $\mathsf{update}(q_s, \tilde{\ell}, E)$
         Step 3: Presenting the result
                 $(\mathsf{Q}_\star, \mathsf{q}_\star, \delta_\star, \mathsf{E}_\star) := (\mathsf{Q}, \mathsf{q}_\star, \delta, \mathsf{E})$
```

Table 8.4: Maximal Fixed Point algorithm for the Pathway Analysis.

associated multiset, $E_\star$, of prefixes exposed by $P_\star$ are added to the otherwise empty Q and E. Then the start state, $\mathsf{q}_\star$, is added to the worklist and the current set of transitions, δ, is set to the empty set.

Line (3) defines the iterative loop, which inspects 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 Subsection 8.3.2 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, $\mathsf{transfer}_{P_\star, \tilde{\ell}}(\mathsf{E}[q_s])$, to the transfer function of Section 8.2.

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 8.3.3. As we shall see, it is crucial for termination that this update is performed in a way such that Q remains finite. In order to ensure this, we shall enable extensive reuse of states, using a clever way of comparing them, and, only if we fail in finding a suitable preexisting state do we add a new one.

8.3.2 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

1. *match* in the sense that one is complementary to the other, and
2. may be *concurrently possible*.

Inspecting the definition of $\mathcal{F}_\star$ for either one of the CFAs of the previous chapters, it becomes clear that $\mathcal{F}_\star$ precisely captures the intuitions suggesting (1) and (2).This leads to estimating the set of enabled transitions simply by taking the pairs in $\mathcal{F}_\star$ where both of the two capabilities are exposed in the present state:

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

Correctness of enabled. The function, $\mathsf{enabled}$, constructed in this way, is *correct* in the sense that it safely over-approximates the set of enabled transitions:

Lemma 8.21 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 6.19 and Lemma 8.5. □

8.3.3 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 corresponding procedure, $\mathsf{update}(q_s, \tilde{\ell}, E)$, is defined in Table 8.5, 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 capabilities; two states, q_1 and q_2, are identified if $\mathsf{dom}(\mathsf{E}[q_1]) = \mathsf{dom}(\mathsf{E}[q_2])$. If a suitable state, q, exists it is used as

```
(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 $\mathsf{Q}$;  $\mathsf{Q} := \mathsf{Q} \cup \{q_t\}$;  $\mathsf{E}[q_t] := \bot_{\mathfrak{M}}$;
(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)  $\mathsf{clean\text{-}up}(\mathsf{Q}, \mathsf{W}, \delta)$
```

Table 8.5: Procedure, $\mathsf{update}(q_s, \tilde{\ell}, E)$, for updating states of pathway automata.

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 a *widening operator*, $\nabla_{\mathfrak{M}} : \mathfrak{M} \times \mathfrak{M} \rightarrow \mathfrak{M}$ (see, e.g., [NNH99]), described below, and q_t is added to the worklist in line (6). The widening operator is defined by

$$(\mathsf{M}_1 \, \nabla_{\mathfrak{M}} \, \mathsf{M}_2)(\ell) = \begin{cases} \mathsf{M}_1(\ell) & \text{if } \mathsf{M}_2(\ell) \leq \mathsf{M}_1(\ell) \\ \mathsf{M}_2(\ell) & \text{if } \mathsf{M}_1(\ell) = 0 \wedge \mathsf{M}_2(\ell) > 0 \\ \infty & \text{otherwise} \end{cases}$$

and guarantees that new information is added to the pre-existing in a manner that stabilises after finitely many iterations and still ensures that $\mathsf{M}_1 \, \mathsf{max}_{\mathfrak{M}} \, \mathsf{M}_2 \leq_{\mathfrak{M}} \mathsf{M}_1 \, \nabla_{\mathfrak{M}} \, \mathsf{M}_2$.

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

Remark 8.22 (Complexity) It is clear that the worst-case complexity of the Pathway Analysis depends on the maximal number of states, as well as the number of transitions leaving each state. Thus, controlling the size of the automaton is one way of controlling the complexity of the algorithm. The partitioning of states, enforced by the check $\mathsf{dom}(\mathsf{E}[q]) = \mathsf{dom}(E)$ in line (1), asserts this kind of control. It is, of course, possible to devise other such *granularity functions* that result in higher or lower complexity and precision.

Another handle to control the complexity is the widening. Here, we interpret

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

Table 8.6: Procedure, $\mathsf{clean\text{-}up}(\mathsf{Q}, \mathsf{W}, \delta)$, for eliminating dead states.

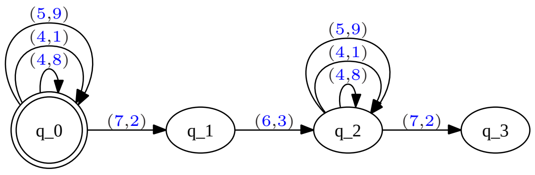
growing numbers of exposed prefixes as a sign of infinite behaviour and go to $\top_{\mathfrak{M}}$ immediately upon detecting such behaviour. One could, of course, make other choices, e.g., in order to separate the first k steps of a recurrent behaviour.

However, in this dissertation we shall not investigate any of these options. ■

Cleaning Up

The cleanup procedure, shown in Table 8.6, simply computes the set of states, Q_{reach}, that is reachable from the start state, $\mathsf{q}_\star$, and uses this to restrict the set of states, Q, the set of transitions, δ, and the worklist, W, by intersection.

Example 8.23 Applying the Pathway Analysis to P_{eat} we obtain the result shown below. Comparing to Fig. 3.28 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 multisets of globally exposed reaction capabilities are the same.



□

8.3.4 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 8.24 (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 Table 8.5 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 Table 8.4 will decrease the size of W by one in every iteration. Eventually W will be empty and hence the algorithm will terminate. $\square$

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

Lemma 8.25 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 Table 8.4. It is maintained due to the construction of δ in line (6) of Table 8.5. $\square$

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

$$P \triangleright E \qquad \text{iff} \qquad \mathcal{E}_\star[\![P]\!] \leq_{\mathfrak{M}} E$$

Using this we can state:

Lemma 8.26 If $\mathsf{PRG}_{\mathbf{C}}(P)$, $P \Rrightarrow Q$ and $P \triangleright E$ then $Q \triangleright E$.

Proof This follows from Corollary 8.5. □

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

Theorem 8.27 The worklist algorithm produces a finite automaton, $(\mathsf{Q}, q_0, \delta, \mathsf{E})$, such that, if

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

then 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 Table 8.4. 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 8.21, and the fact that $\mathsf{enabled}$ 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 Table 8.4. By Theorem 8.18 line (6) then produces E such that $Q \triangleright E$.

Following line (7) of Table 8.4 it is immediate that lines (1-3) of Table 8.5 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 $\mathsf{update}(q, \tilde{\ell}, ...)$. Thus, line (8) of Table 8.6 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 8.25. □

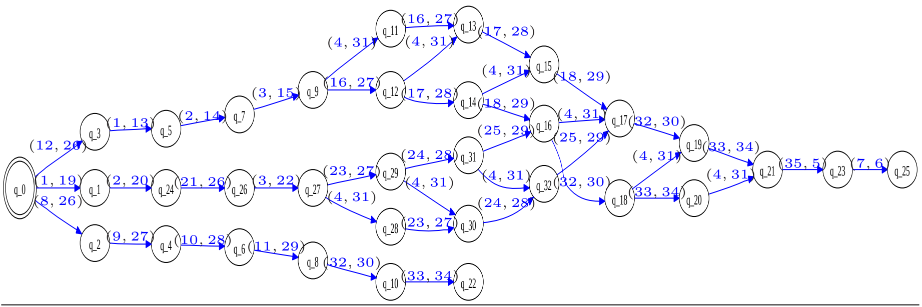


Figure 8.1: Pathway analysis of normal receptor LDL model.

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

$$(\mathsf{q}_\star, \varepsilon, \mathsf{q}_\star) \in \delta^\star \qquad \frac{(\mathsf{q}_\star, \Lambda, q) \in \delta^\star \quad (q, \tilde{\ell}, q') \in \delta}{(\mathsf{q}_\star, \Lambda\tilde{\ell}, q') \in \delta^\star}$$

This allows us to state the following corollary:

Corollary 8.28 *The worklist algorithm produces a finite automaton,* $(\mathsf{Q}, \mathsf{q}_\star, \delta, \mathsf{E})$, *such that, if*

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

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

$$P \triangleright \mathsf{E}[q] \text{ and } (\mathsf{q}_\star, \Lambda, q) \in \delta^\star. \qquad \square$$

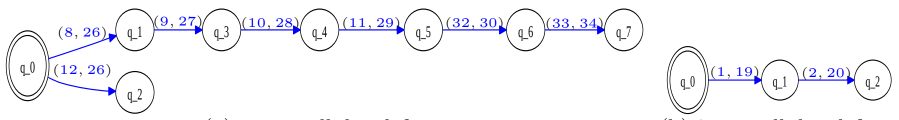
Proof The result follows straightforwardly by induction on the length of Λ. □

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

8.4 CASE: Analysing the LDL Degradation Pathway

When subjecting the normal receptor LDL pathway model to Pathway Analysis we obtain the result shown in Fig. 8.2, independently of the particular CFA used to produce the auxiliary relation $\mathcal{F}_\star$.

The resulting automaton clearly exhibits the transitions that might lead to the configurations identified by the 2CFA analysis. However, some of the exhibited transitions are spurious, false positives that cannot really take place. In



(a) extra-cellular defects (b) intra-cellular defects

Figure 8.2: Pathway analysis of defect receptor LDL models.

particular, only one of the transitions labelled (4, 31), corresponding to *LDL* entering the *XV*, is actually possible, namely the one going from state 16 into state 17. This is because the *EE* must merge with the *LE* before *LDL* can enter the *XV*. Every other occurrence of (4, 31) can be attributed to the fact that the Pathway Analysis incorporates no context information. Consequently, the states 11, 13, 15, 28, 30, and 32 are *analysis artifacts*, i.e., states that are not reachable in practice, and the state 20 is actually a dead end.

8.4.1 Related Diseases

When the LDL pathway model that incorporates exo-plasmic receptor defects is subjected to Pathway Analysis we obtain the pathway automaton shown in Fig. 8.2(a). It is evident that the internalisation of *EE* still works completely as intended. Only the extracellular binding capacity is affected. This result contains no surprises.

In the case of cytosolic defects we obtain the pathway automaton shown in Fig. 8.2(b). Here we see that the receptor protein can ligate an LDL particle, but the steps that correspond to internalisation, i.e., *LDL* entering *EE*, cannot occur. Note in particular, that only the steps associated with the aforementioned modelling artifact might occur. The remaining steps, leading to the rather wild over-approximation of Fig. 7.5(b), cannot occur.

8.5 CASE: Analysing Genetic Transcription

We now turn to the examination of the abstract model of genetic transcription that was presented in Section 3.1.2. When subjecting this model to Pathway Analysis we obtain the pathway automaton shown in Fig. 8.3, regardless of the CFA used for computing $\mathcal{F}_\star$.

The result is rather imprecise, and in more than one way:

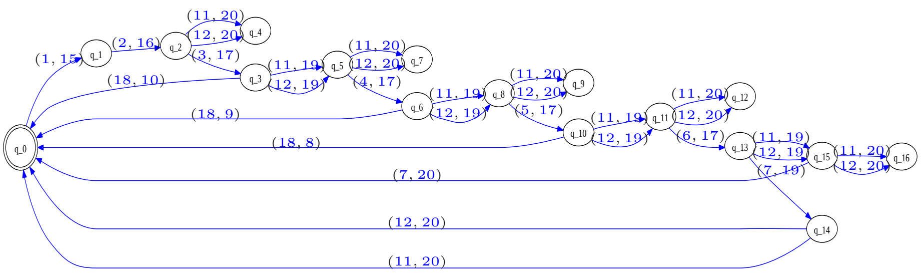


Figure 8.3: Pathway analysis of genetic transcription model.

First of all, the system is not expected to have dead end states. Indeed, inspection reveals that the edges (12, 20), (11, 20), (7, 19) are analysis artifacts, and, if this is taken into account, the model seems quite reasonable.

Secondly, the edges (11, 19), (12, 19), corresponding to the attachment of actual nucleotides, always appear in pairs. This is adequately explained by the fact that the Pathway Analysis does not take the binding of names into account.

8.6 Concluding Remarks

The CFAs of the previous chapters are mainly concerned with the spatial properties of reachable configurations. In contrast, the Pathway Analysis focuses on the causal properties of the realisable transition sequences. Given a model, $P_\star$, the Pathway Analysis computes a finite, partially deterministic, automaton that safely over-approximates the set of, possibly infinite, sequential behaviours that are realisable by $P_\star$.

In aims and scope the Pathway Analysis is superficially related to previous occurrence counting analyses of ambient calculi [HJNN99, LM04, GL05]. However, the Pathway Analysis counts capability prefixes, rather than ambients, and, technically, it resorts to an adaptation of classical Monotone Frameworks [KU77, NNH99], rather than Abstract Interpretation. Thus, the analysis extends a line of work on data flow related analyses for the CCS family of process calculi [NN06, NN08].

Pathway automata model system configurations by extended multisets of exposed prefixes, The dynamic nature of processes is captured by transfer functions in the manner of classical Bit-vector Frameworks. These functions determine how extended multisets grow and shrink as prefixes react. A worklist algorithm computes the final automaton. It expands the state space iteratively until no new states arises by application of the transfer functions.

The practical value of the Pathway Analysis is, in part, determined by its computational complexity. As presented here the algorithm is exponential, but the presented algorithm is remarkably flexible in this respect. Three mechanisms in total are used to control the tradeoff between precision and termination properties of the algorithm:

Firstly, the analysis *counts* and hence a notion of *widening*, in the manner of Abstract Interpretation [CC77, CC79, NNH99], helps to ensure termination in the presence of infinite multiplicities.

Secondly, the $\mathcal{F}_\star$ relation, computed by either of the previously presented CFAs, is used to bound the number of enabled transitions explored by the algorithm.

Finally, an equivalence function on states is used to decide whether a computed state is new or constitutes an update to an already existing state. The only such equivalence explored in this dissertation is equality for $\mathsf{dom}(E)$, but in practice a variety of functions may be used – as long as they are *finitary*, *stable*, and *injective* in the terminology of [NN08]. A fine-grained equivalence is likely to generate fairly precise analysis results at a high complexity. A coarser equivalence relation will give more approximative analysis results at a lower (perhaps even low polynomial) complexity.

Analysis results obtained in the context of the two ongoing case studies are promising, but also indicate viable avenues of improvement:

When applied to the normal receptor LDL pathway model of Section 3.3 the analysis retrieves an automaton that quite clearly identifies the interaction sequences that constitute the pathway. There are a few spurious edges. These analysis artifacts owe to the fact that the analysis does not take context information into account. The results obtained from the defect receptor models are also promising. In the case of exoplasmic defects the resulting pathway automaton basically accounts for the findings of the CFAs. But, in the case of cytosolic defects, the pathway automaton excludes the interactions that might lead to the wild over-approximations observed in the corresponding CFA results. The obtained result is invariant with respect to the CFA used to produce $\mathcal{F}_\star$.

When applied the transcription model of Section 3.4 the analysis retrieves an automaton that exhibits the overall behaviour of the transcription process, i.e. a step-wise elongation procedure that may be interrupted at any point, should the required metabolites not materialise. Again, however, the result contains analysis artifacts that are clearly connected to the fact that the analysis does not take the binding of names into account.

As this model is flat, in the sense that it does not model spatial structure, it is not surprising that the obtained result is also invariant with respect to the CFA used for producing $\mathcal{F}_\star$. This is in contrast to the LDL pathway result, where one might have hoped that 2CFA would lead to better $\mathcal{F}_\star$ estimates.

Technically, the two analysis approaches are complementary — one is context sensitive and considers the bindings of names, while the other is flow-sensitive but ignores bindings and context. This is evident in the results, which seem to indicate that an iterative positive-negative feedback loop between the two analyses would improve the precision of both; an idea that immediately motivates the developments of the following chapter.

Chapter 9

An Iterative Analysis

"It's performance-based. It's systemic. Not just an item here or there. It comes together as a system and operates better. The end is greater than the sum of the parts."

— Jim Folkman

This chapter presents a *iterative analysis* for the BioAmbients language. The analysis is evaluated in the context of both the LDL degradation pathway of Section 3.3 and the transcription model of Section 3.4. None of the presented material has previously been covered.

At first sight the two types of analysis presented in the previous chapters are quite different. The CFA analyses of Chapter 6 and 7 approximate the set of reachable spatial configurations. In contrast, the Pathway Analysis of Chapter 8 approximates the set of realisable causal sequences. These differences, however, are not as profound as they might appear. Clearly, the analyses are all concerned with different aspects of the same thing, namely the run-time behaviour of BioAmbients processes, and, whichever way you put it, this is closely related to the set of reactions that might occur.

The Pathway Analysis already takes avantage of this by using the $\mathcal{F}_\star$ relation, produced by either of the CFA analyses, as a safe approximation to the set of run-time enabled reactions. The Pathway Analysis further refines this first estimate internally, when computing the set of enabled transitions, because in each state only capabilities with a positive number of occurrences can be enabled. This is evident in the Pathway Analysis result for the LDL pathway (Section 8.4), where $\delta_\star$ does not contain all members, $\tilde{\ell}$, of $\mathcal{F}_\star$. As $\delta_\star$ is formally an over-approximation it is safe to conclude that the unused members of $\mathcal{F}_\star$

```
INPUT :    a Flow Logic $\mathcal{FL}$(BioAmbients, ALFP),
           a BioAmbients process $P_\star$, and
           A safe estimate, CP, of the concurrently possible capabilities.

OUTPUT :   an ALFP formula $\varphi$ such that
           $(\mathcal{I},\mathcal{R},\mathcal{F}) \models \varphi \Leftrightarrow (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\top} P_\star$.

METHOD :   Set $\varphi := (\bigwedge\{\mathsf{CP}_\star(\ell_1,\ell_2) \mid (\ell_1,\ell_2) \in \mathsf{CP}\}) \wedge$
                      $(\bigwedge\{\mathbf{C}(n) \mid n \in \mathbf{C}\}) \wedge$
                      $\mathsf{complete}\mathcal{R} \wedge$
                      $(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\top} P$
           while $\varphi$ contains $(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P'$
             and there is a rule $\alpha$ iff $\beta$ in $\mathcal{FL}$
                   and a substitution $\theta$
             such that $\theta\alpha = (\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P'$
            do replace $(\mathcal{I},\mathcal{R},\mathcal{F}) \models^{\mu} P'$ with $\theta\beta$ in $\varphi$.
```

Table 9.1: Parameterised 0CFA clause generator for iteration.

cannot take place.

In the following we take advantage of this observation, and show how a simple iterative analysis can improve on the previous results.

The chapter contains fours sections. In Section 9.1 we present the iterative analysis and state some correctness results. Then, in Section 9.2, we apply it to the LDL pathway model in order to obtain evaluation data. In Section 9.3 we apply it to the transcription model. Finally, in Section 9.4, we summarise our findings.

9.1 Analysis

In the following we shall develop an iterative approach to CFA and Pathway Analysis. In doing so, we pursue the idea that both types of analysis should, of course, only consider the least set of reactions that is known to safely approximate the true set of run-time enabled reactions.

The Pathway Analysis is already parameterised on a safe estimate $\mathcal{F}_\star$ of the run-time enabled reactions, and, in order to specify the iterative analysis, we

```
INPUT :    a BioAmbients process $P_\star$.

OUTPUT :   $(\mathcal{I}_\star, \mathcal{R}_\star, \mathcal{F}_\star)$ and $(\mathsf{Q}_\star, \mathsf{q}_\star, \delta_\star, \mathsf{E}_\star)$ such that
           $(\mathcal{I}_\star, \mathcal{R}_\star, \mathcal{F}_\star) \models^\top P_\star$, $(\mathsf{Q}_\star, \mathsf{q}_\star, \delta_\star, \mathsf{E}_\star) \models \mathsf{pathway}(P_\star)$, and
           $\mathcal{F}_\star = \{\tilde{\ell} \mid \exists q_1, q_2 : (q_1, \tilde{\ell}, q_2) \in \delta_\star\}$

METHOD :   $\varphi_0 := \text{genConstraints}(\text{CFA}, P_\star, \mathsf{CP}_\star)$;
           $(\mathcal{I}, \mathcal{R}, \mathcal{F}) := \text{solve}(\varphi_0)$;
           $(\mathsf{Q}, \mathsf{q}_\star, \delta, \mathsf{E}) := \mathsf{pathway}(P_\star, \mathcal{F})$;
           $\mathsf{CP} := \{\tilde{\ell} \mid \exists q_1, q_2 : (q_1, \tilde{\ell}, q_2) \in \delta\}$;
           while $\mathcal{F} \neq \mathsf{CP}$ do
               $\varphi := \text{genConstraints}(\text{CFA}, P_\star, \mathsf{CP})$;
               $(\mathcal{I}, \mathcal{R}, \mathcal{F}) := \text{solve}(\varphi)$;
               $(\mathsf{Q}, \mathsf{q}_\star, \delta, \mathsf{E}) := \mathsf{pathway}(P_\star, \mathcal{F})$;
               $\mathsf{CP} := \{\tilde{\ell} \mid \exists q_1, q_2 : (q_1, \tilde{\ell}, q_2) \in \delta\}$;
           $(\mathcal{I}_\star, \mathcal{R}_\star, \mathcal{F}_\star) := (\mathcal{I}, \mathcal{R}, \mathcal{F})$;
           $(\mathsf{Q}_\star, \mathsf{q}_\star, \delta_\star, \mathsf{E}_\star) := (\mathsf{Q}, \mathsf{q}_\star, \delta, \mathsf{E})$
```

Table 9.2: Iterative analysis algorithm.

shall parameterise the CFAs in a similar way. The basic idea is to pipe the set of reactions contained in $\delta_\star$ back into the CFA as a refined version of the estimate, $\mathsf{CP}_\star$, of concurrently possible capabilities. This requires a modification of the CFA clause generators. In the case of 0CFA the resulting clause generator is shown in Table 9.1, and the 2CFA can be similarly modified.

In this context the iterative analysis for BioAmbients is defined as shown in Table 9.2, where CFA can be either the 0CFA or the 2CFA. The algorithm starts by computing an ordinary CFA, followed by an ordinary Pathway Analysis, in order to obtain first estimates for $\mathcal{F}$ and CP. If these sets are not equal this computation is repeated until they are.

Lemma 9.1 (Termination) The algorithm terminates.

Proof The CFA ensures that $\mathcal{F} \subseteq \mathsf{CP}$ and the Pathway Analysis ensures that $\{\tilde{\ell} \mid \exists q_1, q_2 : \mathsf{CP} \subseteq \mathcal{F}\}$; hence each iteration either shrinks the estimate of run-time enabled reactions or the algorithm terminates. As $\mathcal{P}(\mathbf{Lab}_\star \times \mathbf{Lab}_\star)$ does not admit infinite descending chains the algorithm will eventally terminate. $\square$

Theorem 9.2 (Soundness) The outputs, $(\mathcal{I}_\star, \mathcal{R}_\star, \mathcal{F}_\star)$ and $(\mathsf{Q}_\star, \mathsf{q}_\star, \delta_\star, \mathsf{E}_\star)$, constitute safe approximations in the manner of Theorems 6.19, 7.30, and 8.28.

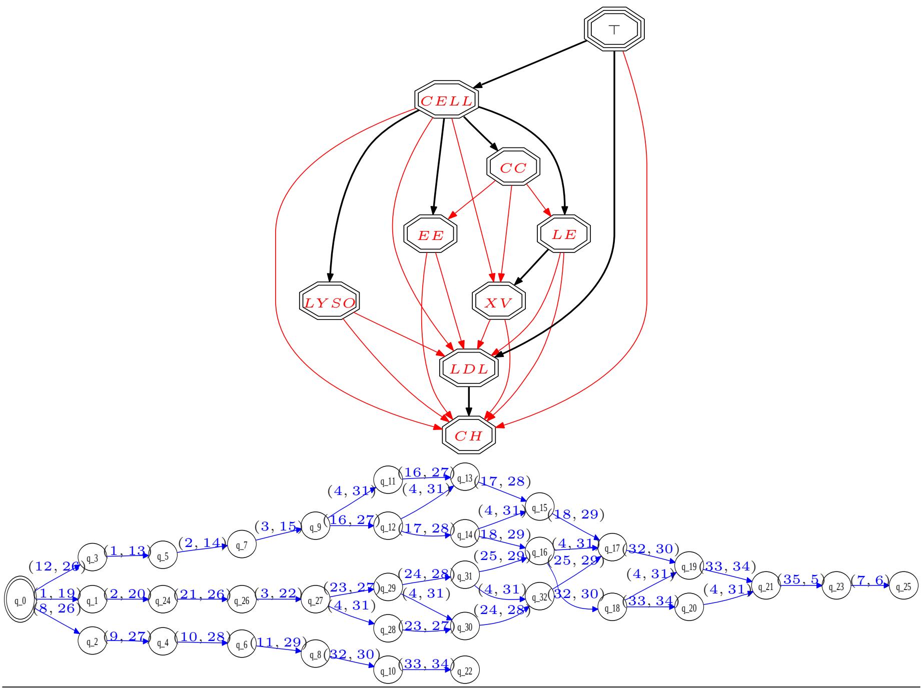


Figure 9.1: Iteration of 0CFA and Pathway Analysis — normal receptors.

Proof This follows directly from the fact that each $\mathcal{F}$ and CP of the descending chain are formally over-approximations. □

9.2 CASE: Analysing the LDL Degradation Pathway

We now conclude our examination of the LDL pathway model by subjecting it to the iterative analyses outlined in the previous section.

We start by applying the 0CFA and the Pathway Analysis iteratively to the normal receptor LDL model. This yields the containment graph and the pathway automaton shown in Fig. 9.1. Here we experience no difference from the results previously obtained by the ordinary 0CFA (Section 6.3) and the ordinary pathway analysis (both based on 0CFA and 2CFA in Section 8.4). The corresponding 0CFA analysis estimates, presented verbatim in Appendix B.1.1

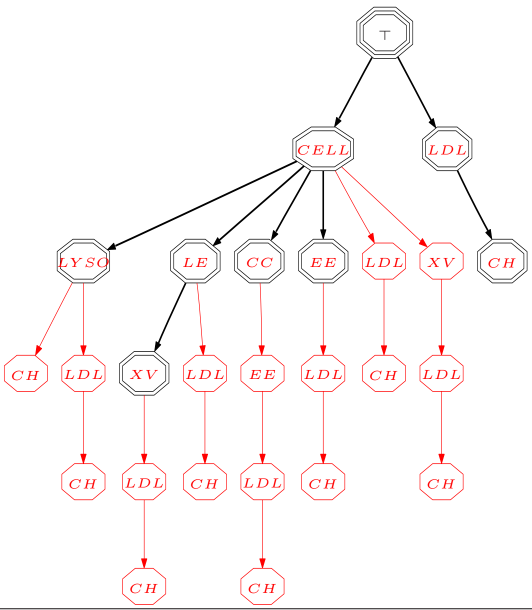


Figure 9.2: Iteration of 2CFA and Pathway Analysis — normal receptors.

and Appendix B.3.1, are identical.

We then go on to apply the 2CFA and the Pathway Analysis iteratively to the normal receptor model. This yields the containment graph shown in Fig. 9.2 and we abstain from showing the related pathway automaton, as it is identical to that shown in Fig. 9.1. Again, there is no observable difference between the shown graph and the previous result for the ordinary 2CFA (Section 7.5). Indeed, the corresponding 2CFA analysis estimates, presented verbatim in Appendix B.2.1 and Appendix B.4.1, are identical.

From these observations we conclude that both the 0CFA and 2CFA results are as precise as possible for this particular model, and, hence, that the 2CFA is vastly more informative than the 0CFA. In the context of the previous examinations this is unsurprising.

9.2.1 Related Diseases

We now turn to revisit the disease related models, turning first to the issue of defect exo-plasmic binding sites. When the corresponding model is subjected to

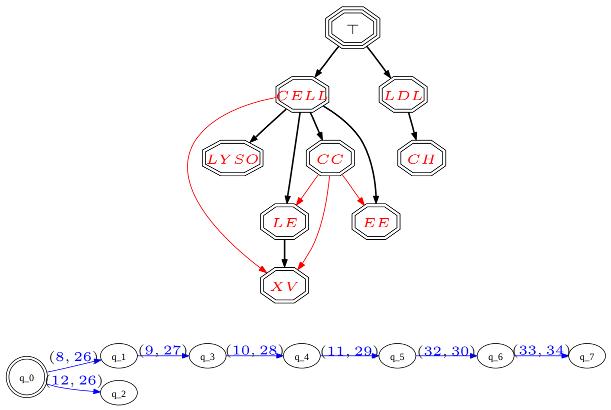


Figure 9.3: Iteration of 0CFA and Pathway Analysis — exo-plasmic defects.

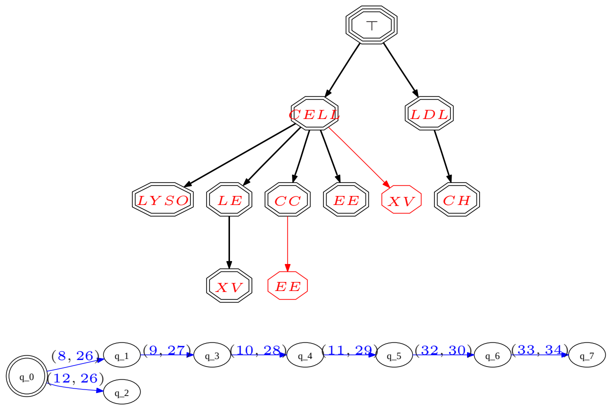


Figure 9.4: Iteration of 2CFA and Pathway Analysis — exo-plasmic defects.

iterative application of 0CFA and Pathway Analysis we obtain the containment graph and pathway automaton shown in Fig. 9.3. At first sight the result looks identical to those previously obtained by the 0CFA (Section 6.3), but inspection of the corresponding analysis results, printed verbatim in Appendices B.1.2 and B.3.2, shows that the number of reactions considered by the iterative 0CFA result is less than half of that considered by the ordinary 0CFA result. As shown in Fig. 9.4, this result basically repeats itself for the 2CFA, also for the verbatim analysis results in Appendices B.2.2 and B.4.2.

We then turn to the model of defect cytosolic binding sites. When this model is subjected to iterative application of 0CFA and Pathway Analysis we obtain the

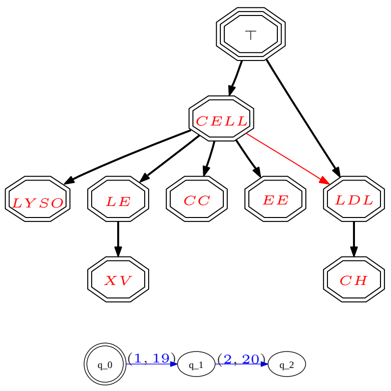


Figure 9.5: Iteration of 0CFA and Pathway Analysis — cytosolic defects.

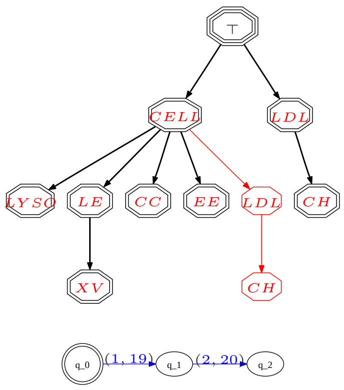


Figure 9.6: Iteration of 2CFA and Pathway Analysis — cytosolic defects.

containment graph and pathway automaton shown in Fig. 9.5. This result shows a remarkable improvement over the result previously obtained by the ordinary 0CFA (Section 6.3). It is now easy to see that cytosolic defects also cripple the system and prevent LDL particles from being internalised. The exo-plasmic domain of the receptor might still ligate LDL particles, but the internal binding sites cannot adhere to the clathrin coat that forms around the coated pit and shapes the early endosome. Hence, the LDL particles cannot be internalised, and the occurrence of LDL in the cytosol is caused by the previously mentioned modelling artifact. The corresponding analysis results can be found in Appendix B.1.3 and B.3.3. Again, the iterative application of 2CFA, shown in Fig. 9.6 exhibits a similar improvement (see Appendix B.2.3 and B.4.3).

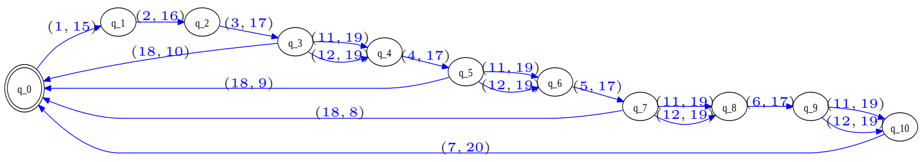


Figure 9.7: Iterative analysis — transcription of single gene.

9.3 CASE: Analysing Genetic Transcription

To complete the examination of the case studies we now return to the transcription model of Section 3.1.3. The result of iterative application of any CFA and Pathway Analysis is the pathway automaton shown in Fig. 9.7.

It is immediate to see that this result is much more precise than the corresponding automaton of the previous chapter. In particular, all of the spurious edges and dead end states have disappeared, and the transcription process is now plain to see; the two first transitions orchestrate the coordination compound comprising the gene and the polymerase. This is followed by four elongation steps, where first the next nucleotide of the sequence is read from the gene (single edge forward), and then either a suitable nucleotide tri-phosphate is ligated (double edge forward, one for each possible nTP) or the transcription is terminated prematurely (backedge to start state). After four such rounds transcription is terminated normally, and the coordination compound is broken (backedge to start state).

The one remaining imprecision is that the analysis cannot distinguish the particular nTP that is ligated in any of the elongation steps. This is no surprise as the Pathway Analysis does not take name bindings into account.

Due to the lack of information regarding name bindings crosstalk is inevitable when, e.g., more than one gene is available for transcription. Hence, we cannot expect the analysis to scale very well. Unfortunately this suspicion is confirmed by the analysis result shown in Fig. 9.8, which is obtained by iterative analysis of a two-gene transcription system. Due to the locking of coordination compounds, we expect the transcription of one gene to terminate, before transcription of the other commences. Thus, intuitively, the corresponding pathway automaton would simply be two separate one-gene automata, but this is not the case.

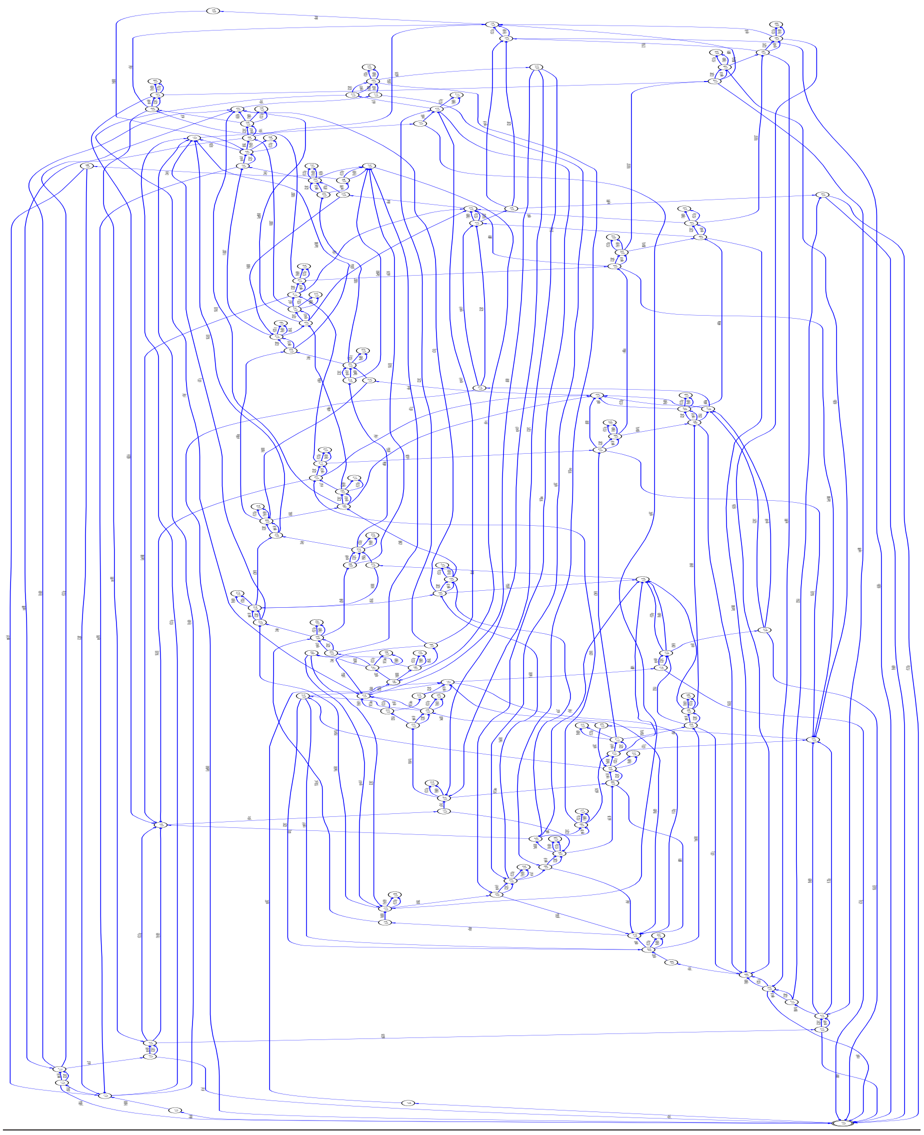


Figure 9.8: Iterative analysis — transcription of two genes. The figure is not intended for reading. Rather, it illustrates the combinatorial blow-up that emerges in the presence of spurious cross-talk between similar subsystems.

9.4 Concluding Remarks

The main contribution of this chapter is a simple idea that, to a large extent, reconciles the otherwise orthogonal analyses of previous chapters. These analyses all take as input a safe estimate of the set of run-time enabled reactions and produce as output a more precise estimate that remains safe. Thus, to obtain the best possible estimate we simply iterate, alternating CFA and Pathway Analysis until the estimate stabilises. This approach is somewhat related to the distinction that is made in Data Flow Analysis between faint and dead variables [NNH99], but, in our case, facilitates analysis refinement rather than program optimisation.

The case studies show that the approach works quite well, in particular for prototypical examples. This positive outcome owes to the profound difference in the approaches of CFA and Pathway Analysis. The former combines context sensitivity with binding information in order to approximate the set of enabled prefixes, whereas the latter relies solely on flow-sensitivity. The outlined iterative approach achieves synergy between these features.

However, the study of genetic transcription also shows that, even for the iterative approach, the Pathway Analysis does not scale in the presence of models that exhibit several similar pathways. Due to the lack of appropriate name binding information the analysis simply cannot distinguish between the pathways. Consequently, the results are obfuscated by cross-talk.

Similarly, the study of the LDL degradation pathway shows that the Pathway Analysis would benefit from the incorporation of context information. The obtained result exhibits several spurious edges corresponding to capabilities reacting ‘out of context’.

In contrast, the iterative CFA results are remarkably precise. Of course, 2CFA is much more informative than 0CFA, but both exhibit good results. And, while CFA does not seem to offer much information about flat models, such as the genetic transcription model, they still prove very useful to the iterative analysis.

Chapter 10

Conclusion

"I never do last words. They always seem so final."
— Laurell K. Hamilton

In this dissertation we have investigated the use of Static Program Analysis techniques in conjunction with BioAmbients process expressions that model sub-cellular biological systems. Our main thesis was that

> The approximative approach of Static Program Analysis can be used to automatically decide biologically relevant properties of process calculus expressions that model biological systems.

Where by *interesting properties* we mean anything that can be used to either pinpoint errors during model development or predict the consequences of perturbations. For the purpose of this dissertation, however, we have focused on the *reachability* of spatial configurations, a property related to the operation of the secretory pathway, and the *realisability* of sequential behaviours, a property related to the operation of cellular pathways in general.

The aim of this chapter is to evaluate our findings with respect to the above thesis. In order to do this we start in Section 10.1 by briefly summing up the work that has been contributed in support of the thesis. In Section 10.2 we gloss over the main findings. Then, in Section 10.3, we discuss how the obtained results reflect upon the thesis and discuss a number of research ideas that would further elaborate on and support the thesis.

10.1 Contributions

In order to support the thesis we have contributed a number of technical developments:

- We have formalised a variant of the BioAmbients language. The proposed variant incorporates a general recursion construct in the manner of the Calculus of Communicating Systems [Mil80] and discards unrestricted non-deterministic choice in favour of guarded sums (Chapter 3).
- The introduction of general recursion complicates many aspects of the technical developments. We have resolved these issues for a class of well-formed initial programs (Chapter 5).
- In order to analyse for structural/spatial properties we have adapted the traditional 0CFA (mono-variant Control Flow Analysis) approach of Flow Logic into a space efficient analysis that for any BioAmbients program, $P_\star$, safely approximates the set of reachable configurations by a single tree (Chapter 6).
- Being dissatisfied with the resulting 0CFA we have made a further adaptation that incorporates context in the manner of 2CFA and relies on multiple auxiliary minor analyses in order to significantly extend the 0CFA approach (Chapter 7).
- To further analyse for causal/temporal properties we have adapted the classical approach of Monotone Frameworks into a ‘Pathway Analysis’ that for any BioAmbients program, $P_\star$, safely approximates the set of realisable transition sequences by a finite automaton (Chapter 8).
- Finally, we have devised an iterative algorithm that, to a certain extent, achieves synergy between the context sensitivity of the Control Flow Analyses and the flow sensitivity of the Pathway Analysis, thereby improving the results of both (Chapter 9).

We have proved that the specified analyses are both exhaustive and correct with respect to BioAmbients programs. Furthermore we have shown that the analyses admit the computation of best analysis results and, hence, that they can be implemented; all results presented throughout the dissertation have been automatically computed by working prototype implementations.

10.2 Evaluation Results

What remains to be discussed is the issue of usefulness, which, in turn, is closely related to the tradeoff between precision and efficiency that is inherently present in all static analyses [NNH99]. Throughout the dissertation we have evaluated each of the analyses by applying it to a number of variations on two small case studies:

Genetic transcription is the process by which genetic information is transcribed into mRNA molecules. The process involves the formation of a temporary coordination compound (complex) that comprises several molecules but operates as a single unit. We have subjected abstract models comprising an RNA polymerase, a supply of nucleotide tri-phosphate, and either a single or two genes to Pathway Analysis (Section 8.5) and iterative analysis (Section 9.3).

The LDL degradation pathway is a prototypical receptor mediated endocytic pathway. The pathway may fail due to defects in the genetic coding of crucial receptor binding sites. High cholesterol and cardiovascular diseases are well-known pathological indications associated with such failure. We have subjected abstract models of both normal and defect systems to control flow (Section 6.3 and Section 7.5), pathway (Section 8.4), and iterative analysis (Section 9.2).

The simplest, and computationally least expensive, analysis is the 0CFA. This simplicity, however, is paid in terms precision. The 0CFA is generally too weak to clearly expose interesting properties in an intelligible manner. In our studies of the LDL degradation pathway (Section 6.3) the analysis does detect serious perturbations, but in all cases the results exhibit many analysis artifacts.

The complexity of the 2CFA analysis is somewhat higher. Correspondingly, the results exhibit very few analysis artifacts and, in many cases, they seem to capture the exact set of reachable spatial configurations (Section 7.5). Somewhat surprisingly, however, the 2CFA does not do significantly better than the 0CFA in pinpointing the effects of receptor related perturbations of the LDL pathway model. Due to the higher precision of the 2CFA, however, we are now able to see that this inadequacy owes to the lack of flow-sensitivity in the CFAs.

The Pathway Analysis, as it is presented in Chapter 8, constitutes another increase in complexity and exhibits an asymptotic worst-case complexity that is exponential in the size of the computed automaton. This complexity is not inherent as particular choices of widening operators and equivalence relations can be used to trade precision for efficiency (Section 8.3.3). The results are quite precise. In particular, the effects of receptor related perturbations of the LDL pathway model are pinpointed very precisely. In fact, the analysis result for the normal LDL pathway model (Section 8.4) is good enough to pinpoint the causal relationship between all of the events depicted in Fig. 2.8 while exhibiting only a few analysis artifacts. These artifacts arise because the analysis does not incorporate context information (Section 8.4). In the case of the gene transcription model the analysis result is also informative, but here we see more analysis artifacts (Section 8.5). These artifacts owe to crosstalk, between seemingly unrelated capability prefixes, that occurs because the analysis does not include name binding information.

These findings are supported by the iterative analysis strategy of Chapter 9. Here the context and name binding information tracked by a CFA analysis influences the Pathway Analysis and, in turn, the flow information tracked by the Pathway Analysis influences the CFA analysis. The improvement is immediate: In the case of receptor related perturbations of the LDL pathway model both of the CFAs now pinpoint the effects very precisely (Section 9.2). And, in the case of genetic transcription, the Pathway Analysis becomes much more precise and exhibits nearly no analysis artifacts (Section 9.3). The iterative Pathway Analysis results for the ordinary LDL pathway model, on the other hand, is not improved. This owes to the very indirect nature of the mutual influence. This also explains the rather poor result obtained for the 2-gene transcription model (Section 9.3), which seems to indicate that, presently, the iterative strategy does not scale well to systems with multiple similar pathways.

10.3 Conclusion and Further Work

In order to finally conclude on the thesis we return to the central question: Can static analysis decide interesting properties of models of biological systems?

Our claim, based on the outlined results, is that yes, it can. The established battery of analyses, which ranges in complexity from (low) polynomial to exponential, is able to accurately, and with increasing precision, pinpoint the most essential aspects of the studied case models.

Based on these results we conjecture that the set of analyses presented in this dissertation constitutes a strong tool that can both support the development of prototypical models and automate significant parts of their post-modelling analysis. Beyond the presented results the former point, in particular, is supported by our, subjective, practical modelling experiences.

Further Work The presented results are very promising. The single sour grape in our basket of attractive fruits is the fact that the analysis package does not maintain the high precision when models grow. Fortunately, the above reflections on the matter reveals a number of key points that may be addressed by further research in order to improve on this:

Flow Sensitive Control Flow Analysis The results clearly indicate that the Control Flow Analyses would benefit from the inclusion of flow-sensitivity [BC05]. In terms of complexity vs. precision, a set of flow sensitive CFAs, independent of the expensive Pathway Analysis, would cover the middle ground between the present CFAs and the Pathway Analysis, and thus be a worthwile extension of the present analysis package. An auxiliary flow-relation, in the vein of the relevant name estimate of Section 7.2, and localised per-label name binding estimates would likely be required.

Name Tracking Pathway Analysis It is also quite clear that the Pathway Analysis would benefit from the inclusion of name binding information. Again, using the CFAs to obtain such information is clearly not the best way. Rather it would be prudent for the Pathway Analysis itself to track the bindings of names, much in the manner of the CFAs. This type of analysis is already being investigated in [NN07], and it seems that an integration with the present Pathway Analysis would be straightforward. In terms of complexity we note that, even though more information is considered in order to make the analysis more precise, the average complexity might well decline as the increased precision would, most likely, lead to fewer transitions in the pathway automaton.

Context Sensitive Pathway Analysis Another way to improve the Pathway Analysis would be to include context information in the manner of e.g. 2CFA. Primarily, this appears to be a necessary precondition for meaningful Pathway Analysis of elaborate cellular systems, where many processes are completely separated by physical barriers. It would also add another dimension to the iterative analysis strategy of Chapter 9 as the contextualised Pathway Analysis and the 2CFA would be able to exchange contextualised, rather than the present flat, information about realisable reactions. However, as a contextualised Pathway Analysis is likely to be computationally quite expensive, the alternative option – incorporating contextualised name binding information directly into the Pathway Analysis – may constitute a more worthwhile pursuit.

Post-analysis Data Mining For large models all of the presented, and indeed all of the above suggested, static analyses are likely to compute quite overwhelming analysis estimates that are beyond human interpretation. Thus it would be obvious to supplement the analyses with a tool-set for post-analysis data-mining. In particular, the idea of using model checking techniques, based on, e.g., Computation Tree Logic, for analysing the always finite pathway automata is quite obvious. A more risky, but potentially interesting, endeavour would be to use model checking related techniques for guiding the computation of the Pathway Analysis, thereby a priori restricting the universe of interest [Saï00]. Dually, the analysis information may be used to direct the model checking of the model itself, which perhaps, in turn, could lead to another iterative approach [BV01].

Stochastic Pathway Analysis Clearly, the Pathway Analysis is somehow related to the state space aggregation algorithms used by the stochastic process algebra community [GHR01, BSW06]. Thus, one could ask if the Pathway Analysis could somehow be used to simplify stochastic models? Well, a direct coupling seems infeasible – the effect of the various approximations on the stochastic information is rather unpredictable. In the vein of the suggestions of the previous paragraph, however, it may be possible to use pathway automata for guiding stochastic model checking [FSCR04].

Other Modelling Languages Finally, one can take a fundamentally different approach and try to influence the precision of the analyses by choosing another modelling formalism. The benefit would be to obtain a cleaner model of, e.g., transport pathways, in the case of Brane calculus, thereby avoiding some analysis artifacts. This is feasible. On the one hand, it would be a large undertaking, and the analysis artifacts would most likely just "move" to another aspects of the model. But on the other hand, a recent study comparing BioAmbients and Brane calculus leads to optimism as the calculi exhibit more similarities than differences [Ver07].

Appendix A

Variants of the LDL Degradation Pathway

A.1 The LDL Pathway with Normal Receptors

LipoProtein =
$[\ LDLrcpt\#!\{ApoB\}^{1}$. enter $ApoB^{2}$. enter ee^{3} . enter xv^{4} . $proc\hat{}?\{Hydr\}^{5}$.
$($ expel $Hydr^{6}$. $\mathbf{0}$
$\mid [$ exit $Hydr^{7}$. $\mathbf{0}\ ]^{CH}\ )\ ]^{LDL}$
Endo =
enter $AP2^{23,16,9}$. exit $AP2^{24,17,10}$. merge− $Le^{25,18,11}$. $\mathbf{0}$
EarlyEndo =
$[$ accept $ee^{22,15}$. Endo $]^{EE}$
ClathrinCoat =
$[\ EErcpt\hat{}?\{ap2\}^{26}$. accept $ap2^{27}$. expel $ap2^{28}$. $\mathbf{0}\ ]^{CC}$
XferVesicle =
$[\ ($ accept xv^{31} . $\mathbf{0}$
$\mid$ exit Le^{32} . merge− $lyso^{33}$. $\mathbf{0}\ )\ ]^{XV}$
LateEndo =
$[\ ($ merge+ Le^{29} . expel Le^{30} . $\mathbf{0}$
$\mid$ XferVesicle $)\ ]^{LE}$
Lysosome =
$[$ merge+ $lyso^{34}$. $proc\_!\{hydr\}^{35}$. $\mathbf{0}\ ]^{LYSO}$
Cell =
$[\ (\ (\ EErcpt\_!\{AP2\}^{8}$. $[$ Endo $]^{EE}$
$+EErcpt\_!\{AP2\}^{12}$. $LDLrcpt\#?\{apob\}^{13}$. accept $apob^{14}$. EarlyEndo
$+LDLrcpt\#?\{apob\}^{19}$. accept $apob^{20}$. $EErcpt\_!\{AP2\}^{21}$. EarlyEndo $)$
$\mid$ ClathrinCoat
$\mid$ LateEndo
$\mid$ Lysosome $)\ ]^{CELL}$

$(LDLrcpt)\,(EErcpt)\,(ApoB)\,(AP2)\,(ee)\,(cc)\,(lyso)\,(xv)$
$(Le)\,(proc)\,(hydr)\,(ap2)\,(ap)$
(LipoProtein $\mid$ Cell)

A.2 The LDL Pathway with Defects in Exoplasmic Domain

LipoProtein =
  [$LDLrcp\#!\{ApoB\}^{1}$. enter $ApoB^{2}$. enter ee^{3} . enter xv^{4} . $proc\hat{\ }?\{Hydr\}^{5}$.
    (expel $Hydr^{6}$. $\mathbf{0}$
      |[exit $Hydr^{7}$. $\mathbf{0}$]^{CH})]^{LDL}
Endo =
  enter $AP2^{23,16,9}$. exit $AP2^{24,17,10}$. merge– $Le^{25,18,11}$. $\mathbf{0}$
EarlyEndo =
  [accept $ee^{22,15}$. Endo]^{EE}
ClathrinCoat =
  [$EErcpt\hat{\ }?\{ap2\}^{26}$. accept $ap2^{27}$. expel $ap2^{28}$. $\mathbf{0}$]^{CC}
XferVesicle =
  [(accept xv^{31} . $\mathbf{0}$
    | exit Le^{32} . merge– $lyso^{33}$. $\mathbf{0}$)]^{XV}
LateEndo =
  [(merge+ Le^{29} . expel Le^{30} . $\mathbf{0}$
    | XferVesicle)]^{LE}
Lysosome =
  [merge+ $lyso^{34}$. $proc\_!\{hydr\}^{35}$. $\mathbf{0}$]^{LYSO}
Cell =
  [(($EErcpt\_!\{AP2\}^{8}$. [Endo]^{EE}
      $+EErcpt\_!\{AP2\}^{12}$. $LDLrcpt\#?\{apob\}^{13}$. accept $apob^{14}$. EarlyEndo
      $+LDLrcpt\#?\{apob\}^{19}$. accept $apob^{20}$. $EErcpt\_!\{AP2\}^{21}$. EarlyEndo)
    | ClathrinCoat
    | LateEndo
    | Lysosome)]^{CELL}

$(LDLrcpt)\,(LDLrcp)\,(EErcpt)\,(ApoB)\,(AP2)\,(ee)\,(cc)\,(lyso)$
  $(xv)\,(Le)\,(proc)\,(hydr)\,(ap2)\,(ap)$
(LipoProtein
  | Cell)

A.3 The LDL Pathway with Defects in Cytosolic Domain

$$\begin{aligned}
&\text{LipoProtein} =\\
&\quad [\ LDLrcpt\#!\{ApoB\}^{1}\,.\,\text{enter } ApoB^{2}\,.\,\text{enter } ee^{3}\,.\,\text{enter } xv^{4}\,.\,proc\hat{}?\{Hydr\}^{5}\,.\\
&\qquad\quad (\ \text{expel } Hydr^{6}\,.\,\mathbf{0}\\
&\qquad\qquad \mid [\ \text{exit } Hydr^{7}\,.\,\mathbf{0}\ ]^{CH}\ )\,]^{LDL}\\
&\text{Endo} =\\
&\quad \text{enter } AP2^{23,16,9}\,.\,\text{exit } AP2^{24,17,10}\,.\,\text{merge}-\ Le^{25,18,11}\,.\,\mathbf{0}\\
&\text{EarlyEndo} =\\
&\quad [\ \text{accept } ee^{22,15}\,.\,\text{Endo}\,]^{EE}\\
&\text{ClathrinCoat} =\\
&\quad [\ EErcp\hat{}?\{ap2\}^{26}\,.\,\text{accept } ap2^{27}\,.\,\text{expel } ap2^{28}\,.\,\mathbf{0}\ ]^{CC}\\
&\text{XferVesicle} =\\
&\quad [\ (\ \text{accept } xv^{31}\,.\,\mathbf{0}\\
&\qquad\ \mid \text{exit } Le^{32}\,.\,\text{merge}-\ lyso^{33}\,.\,\mathbf{0}\ \ )\,]^{XV}\\
&\text{LateEndo} =\\
&\quad [\ (\ \text{merge}+\ Le^{29}\,.\,\text{expel } Le^{30}\,.\,\mathbf{0}\\
&\qquad\ \mid \text{XferVesicle}\ )\,]^{LE}\\
&\text{Lysosome} =\\
&\quad [\ \text{merge}+\ lyso^{34}\,.\,proc\_!\{hydr\}^{35}\,.\,\mathbf{0}\ \ ]^{LYSO}\\
&\text{Cell} =\\
&\quad [\,(\ (\ \ EErcpt\_!\{AP2\}^{8}\,.\,[\ \text{Endo}\,]^{EE}\\
&\qquad\qquad +EErcpt\_!\{AP2\}^{12}\,.\,LDLrcpt\#?\{apob\}^{13}\,.\,\text{accept } apob^{14}\,.\,\text{EarlyEndo}\\
&\qquad\qquad +LDLrcpt\#?\{apob\}^{19}\,.\,\text{accept } apob^{20}\,.\,EErcpt\_!\{AP2\}^{21}\,.\,\text{EarlyEndo}\ )\\
&\qquad\ \mid \text{ClathrinCoat}\\
&\qquad\ \mid \text{LateEndo}\\
&\qquad\ \mid \text{Lysosome}\ )\,]^{CELL}
\end{aligned}$$

$$\begin{aligned}
&(LDLrcpt)\,(EErcpt)\,(EErcp)\,(ApoB)\,(AP2)\,(ee)\,(cc)\,(lyso)\\
&\qquad (xv)\,(Le)\,(proc)\,(hydr)\,(ap2)\,(ap)\\
&(\ \ \text{LipoProtein}\\
&\quad \mid \text{Cell}\ )
\end{aligned}$$

Appendix B

Analysis Results for the LDL Degradation Pathway

B.1 Analysis Results for the 0CFA

B.1.1 The normal LDL Pathway

μ	$\mathcal{I}(\mu)$
$LYSO$	CH, LDL, merge− $lyso^{33}$, exit Le^{32}, accept xv^{31}, merge+ $lyso^{34}$, $proc\_!\{hydr\}^{35}$
XV	CH, LDL, accept xv^{31}, exit Le^{32}, merge− $lyso^{33}$
LE	CH, LDL, merge− Le^{25}, exit $AP2^{24}$, enter $AP2^{23}$, accept ee^{22}, merge− Le^{18}, exit $AP2^{17}$, enter $AP2^{16}$, accept ee^{15}, merge− Le^{11}, exit $AP2^{10}$, enter $AP2^{9}$, merge+ Le^{29}, expel Le^{30}, XV
CC	EE, XV, LE, $EErcpt\hat{}?\{ap2\}^{26}$, accept $ap2^{27}$, expel $ap2^{28}$
EE	CH, LDL, enter $AP2^{9}$, exit $AP2^{10}$, merge− Le^{11}, accept ee^{15}, enter $AP2^{16}$, exit $AP2^{17}$, merge− Le^{18}, accept ee^{22}, enter $AP2^{23}$, exit $AP2^{24}$, merge− Le^{25}
$CELL$	$CH, LDL, EErcpt\_!\{AP2\}^{8}$, $EE, EErcpt\_!\{AP2\}^{12}, LDLrcpt\#?\{apob\}^{13}$, accept $apob^{14}$, $LDLrcpt\#?\{apob\}^{19}$, accept $apob^{20}$, $EErcpt\_!\{AP2\}^{21}, CC, XV$, $LE, LYSO$
CH	exit $Hydr^{7}$
LDL	$LDLrcpt\#!\{ApoB\}^{1}$, enter $ApoB^{2}$, enter ee^{3}, enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH
TOP	$CH, LDL, CELL$

n	$\mathcal{R}(n)$
$ap2$	$AP2$
$apob$	$ApoB$
$Hydr$	$hydr$

ℓ	$\mathcal{F}(\ell)$
33	34
32	30
21	26
12	26
9	27
10	28
11	29
16	27
17	28
18	29
23	27
24	28
25	29
8	26
7	6
35	5
4	31
3	22, 15
2	20, 14
1	19, 13

B.1.2 The LDL Pathway with Defects in Exoplasmic Domain

μ	$\mathcal{I}(\mu)$
$LYSO$	merge− $lyso^{33}$, exit Le^{32}, accept xv^{31}, merge+ $lyso^{34}$, $proc\_!\{hydr\}^{35}$
XV	accept xv^{31}, exit Le^{32}, merge− $lyso^{33}$
LE	merge− Le^{25}, exit $AP2^{24}$, enter $AP2^{23}$, accept ee^{22}, merge− Le^{18}, exit $AP2^{17}$, enter $AP2^{16}$, accept ee^{15}, merge− Le^{11}, exit $AP2^{10}$, enter $AP2^{9}$, merge+ Le^{29}, expel Le^{30}, XV
CC	EE, XV, LE, $EErcpt\hat{}?\{ap2\}^{26}$, accept $ap2^{27}$, expel $ap2^{28}$
EE	enter $AP2^{9}$, exit $AP2^{10}$, merge− Le^{11}, accept ee^{15}, enter $AP2^{16}$, exit $AP2^{17}$, merge− Le^{18}, accept ee^{22}, enter $AP2^{23}$, exit $AP2^{24}$, merge− Le^{25}
$CELL$	$EErcpt\_!\{AP2\}^{8}$, EE, $EErcpt\_!\{AP2\}^{12}$, $LDLrcpt\#?\{apob\}^{13}$, accept $apob^{14}$, $LDLrcpt\#?\{apob\}^{19}$, accept $apob^{20}$, $EErcpt\_!\{AP2\}^{21}$, CC, XV, LE, $LYSO$
CH	exit $Hydr^{7}$
LDL	$LDLrcp\#!\{ApoB\}^{1}$, enter $ApoB^{2}$, enter ee^{3}, enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH
TOP	LDL, $CELL$

n	$\mathcal{R}(n)$
$ap2$	$AP2$

ℓ	$\mathcal{F}(\ell)$
33	34
32	30
21	26
12	26
9	27
10	28
11	29
16	27
17	28
18	29
23	27
24	28
25	29
8	26

B.1.3 The LDL Pathway with Defects in Cytosolic Domain

μ	$\mathcal{I}(\mu)$
$LYSO$	CH, LDL, merge– $lyso^{33}$, exit Le^{32}, accept xv^{31}, merge+ $lyso^{34}$, $proc\_!\{hydr\}^{35}$
XV	CH, LDL, accept xv^{31}, exit Le^{32}, merge– $lyso^{33}$
LE	CH, LDL, merge– Le^{25}, exit $AP2^{24}$, enter $AP2^{23}$, accept ee^{22}, merge– Le^{18}, exit $AP2^{17}$, enter $AP2^{16}$, accept ee^{15}, merge– Le^{11}, exit $AP2^{10}$, enter $AP2^{9}$, merge+ Le^{29}, expel Le^{30}, XV
CC	$EErcp\hat{}?\{ap2\}^{26}$, accept $ap2^{27}$, expel $ap2^{28}$
EE	CH, LDL, enter $AP2^{9}$, exit $AP2^{10}$, merge– Le^{11}, accept ee^{15}, enter $AP2^{16}$, exit $AP2^{17}$, merge– Le^{18}, accept ee^{22}, enter $AP2^{23}$, exit $AP2^{24}$, merge– Le^{25}
$CELL$	$CH, LDL, EErcpt\_!\{AP2\}^{8}$, $EE, EErcpt\_!\{AP2\}^{12}, LDLrcpt\#?\{apob\}^{13}$, accept $apob^{14}$, $LDLrcpt\#?\{apob\}^{19}$, accept $apob^{20}$, $EErcpt\_!\{AP2\}^{21}, CC, XV$, $LE, LYSO$
CH	exit $Hydr^{7}$
LDL	$LDLrcpt\#!\{ApoB\}^{1}$, enter $ApoB^{2}$, enter ee^{3}, enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH
TOP	$CH, LDL, CELL$

n	$\mathcal{R}(n)$
$apob$	$ApoB$
$Hydr$	$hydr$

ℓ	$\mathcal{F}(\ell)$
33	34
32	30
11	29
18	29
25	29
7	6
35	5
4	31
3	22, 15
2	20, 14
1	19, 13

B.2 Analysis Results for the 2CFA

B.2.1 The normal LDL Pathway

μ_{gp}	μ_p	μ	$\mathcal{I}(\mu_{gp}, \mu_p, \mu)$
$CELL$	LE	XV	merge– $lyso^{33}$, exit Le^{32}, accept xv^{31},
			LDL,
$CELL$	LE	LDL	enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$,
			CH,
$CELL$	CC	EE	exit $AP2^{24}$, merge– Le^{25}, exit $AP2^{17}$,
			merge– Le^{18}, exit $AP2^{10}$, merge– Le^{11},
			LDL,
$CELL$	EE	LDL	enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$,
			CH,
$CELL$	XV	LDL	$proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH,
$CELL$	$LYSO$	LDL	$proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH,
$CELL$	LDL	CH	exit $Hydr^{7}$,
$\top$	$CELL$	$LYSO$	$proc\_!\{hydr\}^{35}$, merge+ $lyso^{34}$, accept xv^{31},
			LDL, CH,
$\top$	$CELL$	LE	XV, expel Le^{30}, merge+ Le^{29},
			LDL,
$\top$	$CELL$	CC	expel $ap2^{28}$, accept $ap2^{27}$, $EErcpt\hat{}?\{ap2\}^{26}$,
			EE,
$\top$	$CELL$	EE	merge– Le^{25}, exit $AP2^{24}$, enter $AP2^{23}$,
			accept ee^{22}, merge– Le^{18}, exit $AP2^{17}$,
			enter $AP2^{16}$, accept ee^{15}, merge– Le^{11},
			exit $AP2^{10}$, enter $AP2^{9}$, LDL,
$\top$	$CELL$	XV	accept xv^{31}, merge– $lyso^{33}$, LDL,
$\top$	$CELL$	LDL	enter ee^{3}, enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$,
			expel $Hydr^{6}$, CH,
$\top$	LDL	CH	exit $Hydr^{7}$,
$\top_p$	$\top$	$CELL$	$LYSO$, LE, XV,
			CC, $EErcpt\_!\{AP2\}^{21}$, accept $apob^{20}$,
			$LDLrcpt\#?\{apob\}^{19}$, accept $apob^{14}$,
			$LDLrcpt\#?\{apob\}^{13}$, $EErcpt\_!\{AP2\}^{12}$, EE,
			$EErcpt\_!\{AP2\}^{8}$, LDL,
$\top_p$	$\top$	LDL	CH, expel $Hydr^{6}$, $proc\hat{}?\{Hydr\}^{5}$,
			enter xv^{4}, enter ee^{3}, enter $ApoB^{2}$,
			$LDLrcpt\#!\{ApoB\}^{1}$,
$\top_{gp}$	$\top_p$	$\top$	$CELL$, LDL,
CC	EE	LDL	enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$,
			CH,
LE	XV	LDL	$proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH,
LE	LDL	CH	exit $Hydr^{7}$,
EE	LDL	CH	exit $Hydr^{7}$,
XV	LDL	CH	exit $Hydr^{7}$,
$LYSO$	LDL	CH	exit $Hydr^{7}$,

μ_{gp}	μ_p	μ	p	$\mathcal{R}(\mu_{gp}, \mu_p, \mu, p)$
$\top$	$CELL$	CC	$ap2$	$AP2,$
$\top_p$	$\top$	$CELL$	$apob$	$ApoB,$
$CELL$	$LYSO$	LDL	$Hydr$	$hydr,$
$CELL$	$LYSO$	CH	$Hydr$	$hydr,$
$LYSO$	LDL	CH	$Hydr$	$hydr,$

ℓ	$\mathcal{F}(\ell)$
2	20
2	14
7	6
35	5
4	31
3	22
3	15
1	19
1	13
8	26
9	27
10	28
11	29
16	27
17	28
18	29
23	27
24	28
25	29
12	26
21	26
32	30
33	34

B.2.2 The LDL Pathway with Defects in Exoplasmic Domain

μ_{gp}	μ_p	μ	$\mathcal{I}(\mu_{gp}, \mu_p, \mu)$
$CELL$	LE	XV	merge– $lyso^{33}$, exit Le^{32}, accept xv^{31},
$CELL$	CC	EE	exit $AP2^{24}$, merge– Le^{25}, exit $AP2^{17}$,
			merge– Le^{18}, exit $AP2^{10}$, merge– Le^{11},
$\top$	$CELL$	$LYSO$	$proc\_!\{hydr\}^{35}$, merge+ $lyso^{34}$, accept xv^{31},
$\top$	$CELL$	LE	XV, expel Le^{30}, merge+ Le^{29},
$\top$	$CELL$	CC	expel $ap2^{28}$, accept $ap2^{27}$, $EErcpt\hat{}?\{ap2\}^{26}$,
			EE,
$\top$	$CELL$	EE	merge– Le^{25}, exit $AP2^{24}$, enter $AP2^{23}$,
			accept ee^{22}, merge– Le^{18}, exit $AP2^{17}$,
			enter $AP2^{16}$, accept ee^{15}, merge– Le^{11},
			exit $AP2^{10}$, enter $AP2^{9}$,
$\top$	$CELL$	XV	accept xv^{31}, merge– $lyso^{33}$,
$\top$	LDL	CH	exit $Hydr^{7}$,
$\top_p$	$\top$	$CELL$	$LYSO$, LE, XV,
			CC, $EErcpt\_!\{AP2\}^{21}$, accept $apob^{20}$,
			$LDLrcpt\#?\{apob\}^{19}$, accept $apob^{14}$,
			$LDLrcpt\#?\{apob\}^{13}$, $EErcpt\_!\{AP2\}^{12}$, EE,
			$EErcpt\_!\{AP2\}^{8}$,
$\top_p$	$\top$	LDL	CH, expel $Hydr^{6}$, $proc\hat{}?\{Hydr\}^{5}$,
			enter xv^{4}, enter ee^{3}, enter $ApoB^{2}$,
			$LDLrcp\#!\{ApoB\}^{1}$,
$\top_{gp}$	$\top_p$	$\top$	$CELL$, LDL,

μ_{gp}	μ_p	μ	p	$\mathcal{R}(\mu_{gp}, \mu_p, \mu, p)$
$\top$	$CELL$	CC	$ap2$	$AP2$,

ℓ	$\mathcal{F}(\ell)$
8	26
9	27
10	28
11	29
16	27
17	28
18	29
23	27
24	28
25	29
12	26
21	26
32	30
33	34

B.2.3 The LDL Pathway with Defects in Cytosolic Domain

μ_{gp}	μ_p	μ	$\mathcal{I}(\mu_{gp}, \mu_p, \mu)$
$CELL$	LE	XV	merge– $lyso^{33}$, exit Le^{32}, accept xv^{31},
			LDL,
$CELL$	LE	LDL	enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$,
			CH,
$CELL$	EE	LDL	enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$,
			CH,
$CELL$	XV	LDL	$proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH,
$CELL$	$LYSO$	LDL	$proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH,
$CELL$	LDL	CH	exit $Hydr^{7}$,
$\top$	$CELL$	$LYSO$	$proc\_!\{hydr\}^{35}$, merge+ $lyso^{34}$, accept xv^{31},
			LDL, CH,
$\top$	$CELL$	LE	XV, expel Le^{30}, merge+ Le^{29},
			LDL,
$\top$	$CELL$	CC	expel $ap2^{28}$, accept $ap2^{27}$, $EErcp\hat{}?\{ap2\}^{26}$,
$\top$	$CELL$	EE	merge– Le^{25}, exit $AP2^{24}$, enter $AP2^{23}$,
			accept ee^{22}, merge– Le^{18}, exit $AP2^{17}$,
			enter $AP2^{16}$, accept ee^{15}, merge– Le^{11},
			exit $AP2^{10}$, enter $AP2^{9}$, LDL,
$\top$	$CELL$	XV	accept xv^{31}, merge– $lyso^{33}$, LDL,
$\top$	$CELL$	LDL	enter ee^{3}, enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$,
			expel $Hydr^{6}$, CH,
$\top$	LDL	CH	exit $Hydr^{7}$,
$\top_p$	$\top$	$CELL$	$LYSO$, LE, XV,
			CC, $EErcpt\_!\{AP2\}^{21}$, accept $apob^{20}$,
			$LDLrcpt\#?\{apob\}^{19}$, accept $apob^{14}$,
			$LDLrcpt\#?\{apob\}^{13}$, $EErcpt\_!\{AP2\}^{12}$, EE,
			$EErcpt\_!\{AP2\}^{8}$, LDL,
$\top_p$	$\top$	LDL	CH, expel $Hydr^{6}$, $proc\hat{}?\{Hydr\}^{5}$,
			enter xv^{4}, enter ee^{3}, enter $ApoB^{2}$,
			$LDLrcpt\#!\{ApoB\}^{1}$,
$\top_{gp}$	$\top_p$	$\top$	$CELL$, LDL,
LE	XV	LDL	$proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH,
LE	LDL	CH	exit $Hydr^{7}$,
EE	LDL	CH	exit $Hydr^{7}$,
XV	LDL	CH	exit $Hydr^{7}$,
$LYSO$	LDL	CH	exit $Hydr^{7}$,

μ_{gp}	μ_p	μ	p	$\mathcal{R}(\mu_{gp}, \mu_p, \mu, p)$
$\top_p$	$\top$	$CELL$	$apob$	$ApoB$,
$CELL$	$LYSO$	LDL	$Hydr$	$hydr$,
$CELL$	$LYSO$	CH	$Hydr$	$hydr$,
$LYSO$	LDL	CH	$Hydr$	$hydr$,

ℓ	$\mathcal{F}(\ell)$
2	20
2	14
7	6
35	5
4	31
3	22
3	15
1	19
1	13
11	29
18	29
25	29
32	30
33	34

B.3 Analysis Results for the Iterative 0CFA

B.3.1 The normal LDL Pathway

μ	$\mathcal{I}(\mu)$
$LYSO$	CH, LDL, merge– $lyso^{33}$, exit Le^{32}, accept xv^{31}, merge+ $lyso^{34}$, $proc\_!\{hydr\}^{35}$
XV	CH, LDL, accept xv^{31}, exit Le^{32}, merge– $lyso^{33}$
LE	CH, LDL, merge– Le^{25}, exit $AP2^{24}$, enter $AP2^{23}$, accept ee^{22}, merge– Le^{18}, exit $AP2^{17}$, enter $AP2^{16}$, accept ee^{15}, merge– Le^{11}, exit $AP2^{10}$, enter $AP2^{9}$, merge+ Le^{29}, expel Le^{30}, XV
CC	EE, XV, LE, $EErcpt\hat{}?\{ap2\}^{26}$, accept $ap2^{27}$, expel $ap2^{28}$
EE	CH, LDL, enter $AP2^{9}$, exit $AP2^{10}$, merge– Le^{11}, accept ee^{15}, enter $AP2^{16}$, exit $AP2^{17}$, merge– Le^{18}, accept ee^{22}, enter $AP2^{23}$, exit $AP2^{24}$, merge– Le^{25}
$CELL$	CH, LDL, $EErcpt\_!\{AP2\}^{8}$, EE, $EErcpt\_!\{AP2\}^{12}$, $LDLrcpt\#?\{apob\}^{13}$, accept $apob^{14}$, $LDLrcpt\#?\{apob\}^{19}$, accept $apob^{20}$, $EErcpt\_!\{AP2\}^{21}$, CC, XV, LE, $LYSO$
CH	exit $Hydr^{7}$
LDL	$LDLrcpt\#!\{ApoB\}^{1}$, enter $ApoB^{2}$, enter ee^{3}, enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH
TOP	CH, LDL, $CELL$

ℓ	$\mathcal{F}(\ell)$
33	34
32	30
21	26
12	26
9	27
10	28
11	29
16	27
17	28
18	29
23	27
24	28
25	29
8	26
7	6
35	5
4	31
3	22, 15
2	20, 14
1	19, 13

n	$\mathcal{R}(n)$
$ap2$	$AP2$
$apob$	$ApoB$
$Hydr$	$hydr$

B.3.2 The LDL Pathway with Defects in Exoplasmic Domain

μ	$\mathcal{I}(\mu)$
$LYSO$	merge– $lyso^{33}$, exit Le^{32}, accept xv^{31}, merge+ $lyso^{34}$, $proc\_!\{hydr\}^{35}$
XV	accept xv^{31}, exit Le^{32}, merge– $lyso^{33}$
LE	merge– Le^{25}, exit $AP2^{24}$, enter $AP2^{23}$, accept ee^{22}, merge– Le^{18}, exit $AP2^{17}$, enter $AP2^{16}$, accept ee^{15}, merge– Le^{11}, exit $AP2^{10}$, enter $AP2^{9}$, merge+ Le^{29}, expel Le^{30}, XV
CC	EE, XV, LE, $EErcpt\hat{}?\{ap2\}^{26}$, accept $ap2^{27}$, expel $ap2^{28}$
EE	enter $AP2^{9}$, exit $AP2^{10}$, merge– Le^{11}, accept ee^{15}, enter $AP2^{16}$, exit $AP2^{17}$, merge– Le^{18}, accept ee^{22}, enter $AP2^{23}$, exit $AP2^{24}$, merge– Le^{25}
$CELL$	$EErcpt\_!\{AP2\}^{8}$, EE, $EErcpt\_!\{AP2\}^{12}$, $LDLrcpt\#?\{apob\}^{13}$, accept $apob^{14}$, $LDLrcpt\#?\{apob\}^{19}$, accept $apob^{20}$, $EErcpt\_!\{AP2\}^{21}$, CC, XV, LE, $LYSO$
CH	exit $Hydr^{7}$
LDL	$LDLrcp\#!\{ApoB\}^{1}$, enter $ApoB^{2}$, enter ee^{3}, enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH
TOP	LDL, $CELL$

n	$\mathcal{R}(n)$
$ap2$	$AP2$

ℓ	$\mathcal{F}(\ell)$
33	34
32	30
12	26
9	27
10	28
11	29
8	26

B.3.3 The LDL Pathway with Defects in Cytosolic Domain

μ	$\mathcal{I}(\mu)$
$LYSO$	merge+ $lyso^{34}$, $proc\_!\{hydr\}^{35}$
XV	accept xv^{31}, exit Le^{32}, merge– $lyso^{33}$
LE	merge+ Le^{29}, expel Le^{30}, XV
CC	$EErcp\hat{}?\{ap2\}^{26}$, accept $ap2^{27}$, expel $ap2^{28}$
EE	enter $AP2^{9}$, exit $AP2^{10}$, merge– Le^{11}, accept ee^{15}, enter $AP2^{16}$, exit $AP2^{17}$, merge– Le^{18}, accept ee^{22}, enter $AP2^{23}$, exit $AP2^{24}$, merge– Le^{25}
$CELL$	LDL, $EErcpt\_!\{AP2\}^{8}$, EE, $EErcpt\_!\{AP2\}^{12}$, $LDLrcpt\#?\{apob\}^{13}$, accept $apob^{14}$, $LDLrcpt\#?\{apob\}^{19}$, accept $apob^{20}$, $EErcpt\_!\{AP2\}^{21}$, CC, LE, $LYSO$
CH	exit $Hydr^{7}$
LDL	$LDLrcpt\#!\{ApoB\}^{1}$, enter $ApoB^{2}$, enter ee^{3}, enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH
TOP	LDL, $CELL$

n	$\mathcal{R}(n)$
$apob$	$ApoB$

ℓ	$\mathcal{F}(\ell)$
2	20
1	19

B.4 Analysis Results for the Iterative 2CFA

B.4.1 The normal LDL Pathway

μ_{gp}	μ_p	μ	$\mathcal{I}(\mu_{gp}, \mu_p, \mu)$
$CELL$	LE	XV	merge− $lyso^{33}$, exit Le^{32}, accept xv^{31},
			LDL,
$CELL$	LE	LDL	enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$,
			CH,
$CELL$	CC	EE	exit $AP2^{24}$, merge− Le^{25}, exit $AP2^{17}$,
			merge− Le^{18}, exit $AP2^{10}$, merge− Le^{11},
			LDL,
$CELL$	EE	LDL	enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$,
			CH,
$CELL$	XV	LDL	$proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH,
$CELL$	$LYSO$	LDL	$proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH,
$CELL$	LDL	CH	exit $Hydr^{7}$,
$\top$	$CELL$	$LYSO$	$proc\_!\{hydr\}^{35}$, merge+ $lyso^{34}$, accept xv^{31},
			LDL, CH,
$\top$	$CELL$	LE	XV, expel Le^{30}, merge+ Le^{29},
			LDL,
$\top$	$CELL$	CC	expel $ap2^{28}$, accept $ap2^{27}$, $EErcpt\hat{}?\{ap2\}^{26}$,
			EE,
$\top$	$CELL$	EE	merge− Le^{25}, exit $AP2^{24}$, enter $AP2^{23}$,
			accept ee^{22}, merge− Le^{18}, exit $AP2^{17}$,
			enter $AP2^{16}$, accept ee^{15}, merge− Le^{11},
			exit $AP2^{10}$, enter $AP2^{9}$, LDL,
$\top$	$CELL$	XV	accept xv^{31}, merge− $lyso^{33}$, LDL,
$\top$	$CELL$	LDL	enter ee^{3}, enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$,
			expel $Hydr^{6}$, CH,
$\top$	LDL	CH	exit $Hydr^{7}$,
$\top_p$	$\top$	$CELL$	$LYSO$, LE, XV,
			CC, $EErcpt\_!\{AP2\}^{21}$, accept $apob^{20}$,
			$LDLrcpt\#?\{apob\}^{19}$, accept $apob^{14}$,
			$LDLrcpt\#?\{apob\}^{13}$, $EErcpt\_!\{AP2\}^{12}$, EE,
			$EErcpt\_!\{AP2\}^{8}$, LDL,
$\top_p$	$\top$	LDL	CH, expel $Hydr^{6}$, $proc\hat{}?\{Hydr\}^{5}$,
			enter xv^{4}, enter ee^{3}, enter $ApoB^{2}$,
			$LDLrcpt\#!\{ApoB\}^{1}$,
$\top_{gp}$	$\top_p$	$\top$	$CELL$, LDL,
CC	EE	LDL	enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$,
			CH,
LE	XV	LDL	$proc\hat{}?\{Hydr\}^{5}$, expel $Hydr^{6}$, CH,
LE	LDL	CH	exit $Hydr^{7}$,
EE	LDL	CH	exit $Hydr^{7}$,
XV	LDL	CH	exit $Hydr^{7}$,
$LYSO$	LDL	CH	exit $Hydr^{7}$,

μ_{gp}	μ_p	μ	p	$\mathcal{R}(\mu_{gp}, \mu_p, \mu, p)$
$\top$	$CELL$	CC	$ap2$	$AP2,$
$\top_p$	$\top$	$CELL$	$apob$	$ApoB,$
$CELL$	$LYSO$	LDL	$Hydr$	$hydr,$
$CELL$	$LYSO$	CH	$Hydr$	$hydr,$
$LYSO$	LDL	CH	$Hydr$	$hydr,$

ℓ	$\mathcal{F}(\ell)$
2	20
2	14
7	6
35	5
4	31
3	22
3	15
1	19
1	13
8	26
9	27
10	28
11	29
16	27
17	28
18	29
23	27
24	28
25	29
12	26
21	26
32	30
33	34

B.4.2 The LDL Pathway with Defects in Exoplasmic Domain

μ_{gp}	μ_p	μ	$\mathcal{I}(\mu_{gp}, \mu_p, \mu)$
$CELL$	LE	XV	merge– $lyso^{33}$, exit Le^{32}, accept xv^{31},
$CELL$	CC	EE	exit $AP2^{10}$, merge– Le^{11},
$\top$	$CELL$	$LYSO$	$proc\_!\{hydr\}^{35}$, merge+ $lyso^{34}$, accept xv^{31},
$\top$	$CELL$	LE	XV, expel Le^{30}, merge+ Le^{29},
$\top$	$CELL$	CC	expel $ap2^{28}$, accept $ap2^{27}$, $EErcpt\hat{}?\{ap2\}^{26}$,
			EE,
$\top$	$CELL$	EE	merge– Le^{25}, exit $AP2^{24}$, enter $AP2^{23}$,
			accept ee^{22}, merge– Le^{18}, exit $AP2^{17}$,
			enter $AP2^{16}$, accept ee^{15}, merge– Le^{11},
			exit $AP2^{10}$, enter $AP2^{9}$,
$\top$	$CELL$	XV	accept xv^{31}, merge– $lyso^{33}$,
$\top$	LDL	CH	exit $Hydr^{7}$,
$\top_p$	$\top$	$CELL$	$LYSO$, LE, XV,
			CC, $EErcpt\_!\{AP2\}^{21}$, accept $apob^{20}$,
			$LDLrcpt\#?\{apob\}^{19}$, accept $apob^{14}$,
			$LDLrcpt\#?\{apob\}^{13}$, $EErcpt\_!\{AP2\}^{12}$, EE,
			$EErcpt\_!\{AP2\}^{8}$,
$\top_p$	$\top$	LDL	CH, expel $Hydr^{6}$, $proc\hat{}?\{Hydr\}^{5}$,
			enter xv^{4}, enter ee^{3}, enter $ApoB^{2}$,
			$LDLrcp\#!\{ApoB\}^{1}$,
$\top_{gp}$	$\top_p$	$\top$	$CELL$, LDL,

μ_{gp}	μ_p	μ	p	$\mathcal{R}(\mu_{gp}, \mu_p, \mu, p)$
$\top$	$CELL$	CC	$ap2$	$AP2$,

ℓ	$\mathcal{F}(\ell)$
8	26
9	27
10	28
11	29
12	26
32	30
33	34

B.4.3 The LDL Pathway with Defects in Cytosolic Domain

μ_{gp}	μ_p	μ	$\mathcal{I}(\mu_{gp}, \mu_p, \mu)$
$CELL$	LE	XV	merge− $lyso^{33}$, exit Le^{32}, accept xv^{31},
$CELL$	LDL	CH	exit $Hydr^{7}$,
$\top$	$CELL$	$LYSO$	$proc\_!\{hydr\}^{35}$, merge+ $lyso^{34}$,
$\top$	$CELL$	LE	XV, expel Le^{30}, merge+ Le^{29},
$\top$	$CELL$	CC	expel $ap2^{28}$, accept $ap2^{27}$, $EErcp\hat{}?\{ap2\}^{26}$,
$\top$	$CELL$	EE	merge− Le^{25}, exit $AP2^{24}$, enter $AP2^{23}$,
			accept ee^{22}, merge− Le^{18}, exit $AP2^{17}$,
			enter $AP2^{16}$, accept ee^{15}, merge− Le^{11},
			exit $AP2^{10}$, enter $AP2^{9}$,
$\top$	$CELL$	LDL	enter ee^{3}, enter xv^{4}, $proc\hat{}?\{Hydr\}^{5}$,
			expel $Hydr^{6}$, CH,
$\top$	LDL	CH	exit $Hydr^{7}$,
$\top_p$	$\top$	$CELL$	$LYSO, LE, CC$,
			$EErcpt\_!\{AP2\}^{21}$, accept $apob^{20}$, $LDLrcpt\#?\{apob\}^{19}$,
			accept $apob^{14}$, $LDLrcpt\#?\{apob\}^{13}$, $EErcpt\_!\{AP2\}^{12}$,
			EE, $EErcpt\_!\{AP2\}^{8}$, LDL,
$\top_p$	$\top$	LDL	CH, expel $Hydr^{6}$, $proc\hat{}?\{Hydr\}^{5}$,
			enter xv^{4}, enter ee^{3}, enter $ApoB^{2}$,
			$LDLrcpt\#!\{ApoB\}^{1}$,
$\top_{gp}$	$\top_p$	$\top$	$CELL, LDL$,

μ_{gp}	μ_p	μ	p	$\mathcal{R}(\mu_{gp}, \mu_p, \mu, p)$
$\top_p$	$\top$	$CELL$	$apob$	$ApoB$,

ℓ	$\mathcal{F}(\ell)$
2	20
1	19

Appendix C

Analysis Results for Genetic Transcription

C.1 Ordinary 2CFA - One Gene

μ_{gp}	μ_p	μ	$\mathcal{I}(\mu_{gp},\mu_p,\mu)$
$\top_{gp}$	$\top_p$	$\top$	$MRNA, b?\{dp\}^{20}, p?\{dp\}^{19},$
			$b!\{d\}^{18}, gene?\{p\}^{17}, gene?\{b\}^{16},$
			$tr?\{gene\}^{15}, t!\{t\}^{14}, g!\{g\}^{13},$
			$c!\{c\}^{12}, a!\{a\}^{11}, b1?\{d11\}^{10},$
			$b1?\{d11\}^{9}, b1?\{d11\}^{8}, b1!\{d1\}^{7},$
			$g1!\{c\}^{6}, g1!\{a\}^{5}, g1!\{c\}^{4},$
			$g1!\{a\}^{3}, g1!\{b1\}^{2}, tr!\{g1\}^{1},$

μ_{gp}	μ_p	μ	p	$\mathcal{R}(\mu_{gp},\mu_p,\mu,p)$
$\top_{gp}$	$\top_p$	$\top$	$gene$	$g1,$
$\top_{gp}$	$\top_p$	$\top$	b	$c, a, b1,$
$\top_{gp}$	$\top_p$	$\top$	dp	$c, a, d1,$
$\top_{gp}$	$\top_p$	$\top$	p	$c, a, b1,$
$\top_{gp}$	$\top_p$	$\top$	$d11$	$d,$

ℓ	$\mathcal{F}(\ell)$
18	8
18	9
18	10
7	19
7	20
2	17
2	16
3	17
3	16
4	17
4	16
11	19
11	20
5	17
5	16
12	19
12	20
6	17
6	16
1	15

C.2 Iterative 2CFA - One Gene

μ_{gp}	μ_p	μ	$\mathcal{I}(\mu_{gp}, \mu_p, \mu)$
$\top_{gp}$	$\top_p$	$\top$	$MRNA, b?\{dp\}^{20}, p?\{dp\}^{19},$
			$b!\{d\}^{18}, gene?\{p\}^{17}, gene?\{b\}^{16},$
			$tr?\{gene\}^{15}, t!\{t\}^{14}, g!\{g\}^{13},$
			$c!\{c\}^{12}, a!\{a\}^{11}, b1?\{d11\}^{10},$
			$b1?\{d11\}^{9}, b1?\{d11\}^{8}, b1!\{d1\}^{7},$
			$g1!\{c\}^{6}, g1!\{a\}^{5}, g1!\{c\}^{4},$
			$g1!\{a\}^{3}, g1!\{b1\}^{2}, tr!\{g1\}^{1},$

μ_{gp}	μ_p	μ	p	$\mathcal{R}(\mu_{gp}, \mu_p, \mu, p)$
$\top_{gp}$	$\top_p$	$\top$	$gene$	$g1,$
$\top_{gp}$	$\top_p$	$\top$	p	$c, a,$
$\top_{gp}$	$\top_p$	$\top$	dp	$c, a, d1,$
$\top_{gp}$	$\top_p$	$\top$	b	$b1,$
$\top_{gp}$	$\top_p$	$\top$	$d11$	$d,$

ℓ	$\mathcal{F}(\ell)$
18	8
18	9
18	10
7	20
2	16
3	17
4	17
11	19
5	17
12	19
6	17
1	15

C.3 Ordinary 2CFA - Two Genes

μ_{gp}	μ_p	μ	$\mathcal{I}(\mu_{gp}, \mu_p, \mu)$
$\top_{gp}$	$\top_p$	$\top$	$MRNA, b?\{dp\}^{30}, p?\{dp\}^{29},$
			$b!\{d\}^{28}, gene?\{p\}^{27}, gene?\{b\}^{26},$
			$tr?\{gene\}^{25}, t!\{t\}^{24}, g!\{g\}^{23},$
			$c!\{c\}^{22}, a!\{a\}^{21}, b2?\{d21\}^{20},$
			$b2?\{d21\}^{19}, b2?\{d21\}^{18}, b2!\{d2\}^{17},$
			$g2!\{c\}^{16}, g2!\{a\}^{15}, g2!\{c\}^{14},$
			$g2!\{a\}^{13}, g2!\{b2\}^{12}, tr!\{g2\}^{11},$
			$b1?\{d11\}^{10}, b1?\{d11\}^{9}, b1?\{d11\}^{8},$
			$b1!\{d1\}^{7}, g1!\{c\}^{6}, g1!\{a\}^{5},$
			$g1!\{c\}^{4}, g1!\{a\}^{3}, g1!\{b1\}^{2},$
			$tr!\{g1\}^{1},$

μ_{gp}	μ_p	μ	p	$\mathcal{R}(\mu_{gp}, \mu_p, \mu, p)$
$\top_{gp}$	$\top_p$	$\top$	$gene$	$g2, g1,$
$\top_{gp}$	$\top_p$	$\top$	b	$c, a, b2, b1,$
$\top_{gp}$	$\top_p$	$\top$	dp	$c, a, d2, d1,$
$\top_{gp}$	$\top_p$	$\top$	p	$c, a, b2, b1,$
$\top_{gp}$	$\top_p$	$\top$	$d21$	$d,$
$\top_{gp}$	$\top_p$	$\top$	$d11$	$d,$

ℓ	$\mathcal{F}(\ell)$
7	29
7	30
2	27
2	26
3	27
3	26
4	27
4	26
5	27
5	26
6	27
6	26
1	25
28	8
28	9
28	10
28	18
28	19
28	20
17	29
17	30
12	27
12	26
13	27
13	26
14	27
14	26
21	29
21	30
15	27
15	26
22	29
22	30
16	27
16	26
11	25

C.4 Iterative 2CFA - Two Genes

μ_{gp}	μ_p	μ	$\mathcal{I}(\mu_{gp}, \mu_p, \mu)$
$\top_{gp}$	$\top_p$	$\top$	$MRNA, b?\{dp\}^{30}, p?\{dp\}^{29},$
			$b!\{d\}^{28}, gene?\{p\}^{27}, gene?\{b\}^{26},$
			$tr?\{gene\}^{25}, t!\{t\}^{24}, g!\{g\}^{23},$
			$c!\{c\}^{22}, a!\{a\}^{21}, b2?\{d21\}^{20},$
			$b2?\{d21\}^{19}, b2?\{d21\}^{18}, b2!\{d2\}^{17},$
			$g2!\{c\}^{16}, g2!\{a\}^{15}, g2!\{c\}^{14},$
			$g2!\{a\}^{13}, g2!\{b2\}^{12}, tr!\{g2\}^{11},$
			$b1?\{d11\}^{10}, b1?\{d11\}^{9}, b1?\{d11\}^{8},$
			$b1!\{d1\}^{7}, g1!\{c\}^{6}, g1!\{a\}^{5},$
			$g1!\{c\}^{4}, g1!\{a\}^{3}, g1!\{b1\}^{2},$
			$tr!\{g1\}^{1},$

μ_{gp}	μ_p	μ	p	$\mathcal{R}(\mu_{gp}, \mu_p, \mu, p)$
$\top_{gp}$	$\top_p$	$\top$	$gene$	$g2, g1,$
$\top_{gp}$	$\top_p$	$\top$	b	$c, a, b2, b1,$
$\top_{gp}$	$\top_p$	$\top$	dp	$c, a, d2, d1,$
$\top_{gp}$	$\top_p$	$\top$	p	$c, a, b2, b1,$
$\top_{gp}$	$\top_p$	$\top$	$d21$	$d,$
$\top_{gp}$	$\top_p$	$\top$	$d11$	$d,$

ℓ	$\mathcal{F}(\ell)$
7	29
7	30
2	27
2	26
3	27
3	26
4	27
4	26
5	27
5	26
6	27
6	26
1	25
28	8
28	9
28	10
28	18
28	19
28	20
17	29
17	30
12	27
12	26
13	27
13	26
14	27
14	26
21	29
21	30
15	27
15	26
22	29
22	30
16	27
16	26
11	25

Bibliography

[AJL$^+$02] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. *Molecular Biology of The Cell*. Garland Science, 4th edition, 2002.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[BB90] Gerard Berry and Gerard Boudol. The chemical abstract machine. In *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 81–94, New York, NY, USA, 1990. ACM Press.

[BC05] Chiara Braghin and Agostino Cortesi. Flow-sensitive leakage analysis in mobile ambients. *Electr. Notes Theor. Comput. Sci.*, 128(5):17–25, 2005.

[BCP06] Ralf Blossey, Luca Cardelli, and Andrew Phillips. A compositional approach to the stochastic dynamics of gene networks. *Transactions in Computational Systems Biology*, 3939:99–122, 2006.

[BDNN98] Chiara Bodei, Pier Paolo Degano, Flemming Nielson, and Hanne Riis Nielson. Control flow analysis for the π-calculus. In *CONCUR 1998 – Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 1998.

[BDNN01] Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static analysis for secrecy and non-interference in networks of processes. In *Proceedings of Parallel Computing Technologies*, volume 2127 of *Lecture Notes in Computer Science*, pages 27–41, 2001.

[BDPZ03] Chiara Bodei, Pier Paolo Degano, Corrado Priami, and Nicola Zannone. An enhanced cfa for security policies. In *Proc. of the Workshop on Issues on the Theory of Security (WITS'03)*, 2003.

[BSW06] Hauke Busch, Werner Sandmann, and Verena Wolf. A numerical aggregation algorithm for the enzyme-catalyzed substrate conversion. In Corrado Priami, editor, *Proc. of Computational Methods in Systems Biology (CMSB'06)*, volume 4210 of *Lecture Notes in Computer Science*, pages 298–311. Springer, 2006.

[BV01] Guillaume Brat and Willem Visser. Combining static analysis and model checking for software analysis. In *Proc. of Proceedings of the 16th IEEE international conference on Automated software engineering (ASE'01)*, page 262, Washington, DC, USA, 2001. IEEE Computer Society.

[Cai04] Luis Caires. Behavioral and spatial observations in a logic for the pi-calculus. In Igor Walukiewicz, editor, *Proceedings of 7th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'04)*, volume 2987 of *Lecture Notes in Computer Science*. Springer, 2004.

[Car04] Luca Cardelli. Bioware languages. In Andrew Herbert and Karen S. Jones, editors, *Computer Systems: Theory, Technology, and Applications - A Tribute to Roger Needham*, Monographs in Computer Science, pages 59–65. Springer, 2004.

[Car05a] Luca Cardelli. Abstract machines of systems biology. *Transactions on Computational Systems Biology*, III:145–168, 2005.

[Car05b] Luca Cardelli. Brane calculi. In Vincent Danos and Vincent Schachter, editors, *Proc. of Computational Methods in Systems Biology (CMSB'04)*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–280. Springer, 2005.

[CC77] Patrick Cousot and Radhia Cousot. Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL'1977: Proc. of the 4th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 238–252. ACM Press, 1977.

[CC79] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *POPL'1979: Proc. of the 6th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 269–282. ACM Press, 1979.

[CDGH06] Muffy Calder, Adam Duguid, Stephen Gilmore, and Jane Hillston. Stronger computational modelling of signalling pathways using both continuous and discrete-state methods. In Corrado Priami, editor, *Proc. of Computational Methods in Systems Biology (CMSB'06)*, volume 4210 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2006.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[CGH05] Muffy Calder, Stephen Gilmore, and Jane Hillston. Automatically deriving ODEs from process algebra models of signalling pathways. In Gordon Plotkin, editor, *Proc. of Computational Methods in Systems Biology (CMSB'05)*, pages 204–215, April 2005.

[CGH06] Muffy Calder, Stephen Gilmore, and Jane Hillston. Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. In *Transactions on Computational Systems Biology VII*, volume 4230 of *Lecture Notes in Computer Science*. Springer, 1–23 2006.

[CGHV06] Muffy Calder, Stephen Gilmore, Jane Hillston, and Vladislav Vyshemirsky. Formal methods for biochemical signalling pathways. In *BCS*, 2006. To appear.

[CGP00] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, January 2000.

[CP05] Luca Cardelli and Sylvain Pradalier. Where membranes meet complexes. In *Proc. of BioConcur'05*, 2005.

[Cri58] Francis Henry Compton Crick. The biological replication of macromolecules. In *Proceedings of the 12th Symposia of the Society for Experimental Biology*, pages 138–163, New York, 1958. Academic Press.

[DK03] Vincent Danos and Jean Krivine. Formal molecular biology done in CCS-R. In *Proc. of BioConcur'03*, Electronic Notes in Theoretical Computer Science. Elsevier, 2003.

[DK04] Vincent Danos and Jean Krivine. Reversible communicating systems. In *CONCUR 2004 – Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 292–307. Springer, September 2004.

[DL03a] Vincent Danos and Cosimo Laneve. Core formal molecular biology. In *Proc. of the 12th ESOP*, volume 2618 of *Lecture Notes in Computer Science*, pages 308–318. Springer, 2003.

[DL03b] Vincent Danos and Cosimo Laneve. Graphs for core molecular biology. In *Proc. of Computational Methods in Systems Biology (CMSB'03)*, volume 2602 of *Lecture Notes in Computer Science*, pages 34–46. Springer, 2003.

[DL04] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, 2004.

[DP02] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, April 2002.

[DP04] Vincent Danos and Sylvain Pradalier. Projective brane calculus. In *Proc. of Computational Methods in Systems Biology (CMSB'04)*, 2004.

[FSCR04] Francois Fages, Sylvain Sollman, and Nathalie Chabrier-Rivier. Modelling and querying interaction networks in the biochemical abstract machine biocham. *Journal of Biological Physics and Chemistry*, 4:64–73, 2004.

[GHR01] Stephen Gilmore, Jane Hillston, and Marina Ribaudo. An efficient algorithm for aggregating PEPA models. *Software Engineering*, 27(5):449–464, 2001.

[Gil76] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22:403–434, 1976.

[Gil77] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[GL05] Roberta Gori and Francesca Levi. A new occurrence counting analysis for bioambients. In *Proceedings of Asian Symposium on Programming Languages and Systems*, volume 3780 of *Lecture Notes in Computer Science*, pages 381–400, 2005.

[Her02] Holger Hermanns. *Interactive Markov Chains and the Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2002.

[Hil96] Jane Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, 1996.

[HJNN99] René R. Hansen, Jacob G. Jensen, Flemming Nielson, and Hanne Riis Nielson. Abstract interpretation of Mobile Ambients. In *SAS'99: Proc. of the 6th International Static Analysis Symposium*, volume 1694 of *Lecture Notes in Computer Science*, pages 135–148, 1999.

[Jon07] Mike Jones. Flow of information in biological systems. Original Work by Mike Jones for Wikipedia, January 2007. Licensed under the Creative Commons Attribution ShareAlike License v. 2.5: http://creativecommons.org/licenses/by-sa/2.5.

[JW95] Suresh Jagannathan and Stephen Weeks. A unified treatment of flow analysis in higher-order languages. In *POPL '95: Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 393–407, New York, NY, USA, 1995. ACM Press.

[Kil72] Gary Arlen Kildall. *Global expression optimization during compilation*. PhD thesis, 1972.

[Kit02] Hiroaki Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, March 2002.

[KN06] Céline Kuttler and Joachim Niehren. Gene regulation in the pi-calculus: simulating cooperativity at the lambda switch. *Transactions on Computational Systems Biology VII*, 4230:24–55, 2006.

[KU77] John B. Kam and Jeffrey D. Ullman. Monotone Data Flow Analysis frameworks. *Acta Informatica*, 7(3):305–317, 1977.

[Kut07] Céline Kuttler. *Modeling bacterial gene expression in a stochastic pi-calculus with concurrent objects*. PhD thesis, University of Lille 1, 2007.

[LBZ$^+$99] Harvey Lodish, Arnold Berk, S. Lawrence Zipursky, Paul Matsudaira, David Baltimore, and James E. Darnell. *Molecular Cell Biology*. W.H. Freeman and Company, 4th edition, 1999. Fig. **??** reprinted with the permission of the publisher.

[LM01] Francesca Levi and Sergio Maffeis. An abstract interpretation framework for analysing Mobile Ambients. In *SAS'2001: Proc. of the 8th International Static Analysis Symposium*, volume 2126 of *Lecture Notes in Computer Science*, pages 395–411. Springer, 2001.

[LM04] Francesca Levi and Sergio Maffeis. On abstract interpretation of mobile ambients. *Information and Computation*, 188(2):179–240, 2004.

[LS03] Francesca Levi and Davide Sangiorgi. Mobile safe ambients. *ACM Transactions on Programming Languages and Systems (TOPLAS'03)*, 25(1):1–69, 2003.

[Mil78] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, December 1978.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[Mil99] Robin Milner. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of Mobile processes (I and II). *Information and Computation*, 100(1):1–77, 1992.

[NN97] Hanne Riis Nielson and Flemming Nielson. Infinitary Control Flow Analysis: a collecting semantics for closure analysis. In *POPL'1997: Proc. of the 24th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 332–345, 1997.

[NN98] Hanne Riis Nielson and Flemming Nielson. Flow logics for constraint based analysis. In *Proc. of the 7th International Conference on Compiler Construction (CC'98)*, volume 1383 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 1998.

[NN00] Hanne Riis Nielson and Flemming Nielson. Shape analysis for mobile ambients. In *POPL'00: Proc. of the 27th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 142–154. ACM Press, 2000.

[NN02] Flemming Nielson and Hanne Riis Nielson. Flow Logic: a multi-paradigmatic approach to static analysis. In *The Essense of Computation: Complexity, Analysis, Transformation*, volume 2566 of *Lecture Notes in Computer Science*, pages 223–244. Springer, 2002.

[NN06] Hanne Riis Nielson and Flemming Nielson. Data Flow Analysis for CCS. In Thomas Reps and Mooly Sagiv, editors, *Festschrift for Reinhard Wilhelm*, volume 4444 of *Lecture Notes in Computer Science*. Springer, 2006.

[NN07] Hanne Riis Nielson and Flemming Nielson. A flow-sensitive analysis of privacy properties. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, 2007. To Appear.

[NN08] Hanne Riis Nielson and Flemming Nielson. A Monotone Framework for CCS. *under revision for Computer Languages, Systems & Structures*, 2008.

[NNB04] Hanne Riis Nielson, Flemming Nielson, and Mikael Buchholtz. Security for mobility. In *FOSAD 2001/2002 Tutorial Lecture Notes*, volume 2946 of *Lecture Notes in Computer Science*, pages 207–265. Springer, 2004.

[NNH99] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer, 1999.

[NNH02] Flemming Nielson, Hanne Riis Nielson, and René R. Hansen. Validating firewalls using Flow Logics. *Theoretical Computer Science*, 283(2):381–418, 2002.

[NNHJ99] Flemming Nielson, Hanne Riis Nielson, René R. Hansen, and Jacob G. Jensen. Validating firewalls in Mobile Ambients. In *CONCUR 1999 – Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 463–477. Springer, 1999.

[NNP04] Hanne Riis Nielson, Flemming Nielson, and Henrik Pilegaard. Spatial analysis of BioAmbients. In *SAS'04: Proc. of the 11th International Static Analysis Symposium*, volume 3148 of *Lecture Notes in Computer Science*, pages 69–83, 2004.

[NNP07] Flemming Nielson, Hanne Riis Nielson, and Henrik Pilegaard. What is a free name in a process algebra? *Information Processing Letters*, 103(5):188–194, 2007.

[NNPdR07] Flemming Nielson, Hanne Riis Nielson, Corrado Priami, and Debora S. da Rosa. Control flow analysis for BioAmbients. In *Proceedings of the First Workshop on Concurrent Models in Molecular Biology (BioConcur 2003)*, volume 180 of *Electronic Notes in Theoretical Computer Science*, pages 65–79, 2007., 2007.

[NNS02] Flemming Nielson, Hanne Riis Nielson, and Helmuth Seidl. A succinct solver for ALFP. *Nordic Journal of Computing*, 9:335–372, 2002.

[NNS$^+$04] Flemming Nielson, Hanne Riis Nielson, Hongyan Sun, Mikael Buchholtz, René Rydhof Hansen, Henrik Pilegaard, and Helmuth Seidl. The succinct solver suite. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 2004.

[Par01] Joachim Parrow. An introduction to the π-calculus. In Bergstra, Ponse, and Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001.

[PC04] Andrew Phillips and Luca Cardelli. A correct abstract machine for the stochastic pi-calculus. In *Bioconcur'04*. Electronic Notes in Theoretical Computer Science, August 2004.

[PC05] Andrew Phillips and Luca Cardelli. A graphical representation for the stochastic pi-calculus. In *Proc. of Bioconcur'05*, August 2005.

[Plo04] Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[PNN05] Henrik Pilegaard, Flemming Nielson, and H. R. Nielson. Static analysis of a model of the LDL degradation pathway. In Gordon D. Plotkin, editor, *Proc. of Computational Methods in Systems Biology (CMSB'05)*. University of Edinburgh, April 2005.

[PNN06a] Henrik Pilegaard, Flemming Nielson, and Hane Riis Nielson. Active evaluation contexts for process algebra. In *Proceedings of Structural Operational Semantics*, 2006. To appear.

[PNN06b] Henrik Pilegaard, Flemming Nielson, and Hanne Riis Nielson. Context dependent analysis of bioambients. In Francesco Ranzato, editor, *Proceedings of the 1st International Workshop on Emerging Applications of Abstract Interpretation*, 2006.

[PNN07] Henrik Pilegaard, Flemming Nielson, and Hanne Riis Nielson. Pathway analysis for bioambients. Submitted for publication, 2007.

[PQ04] Corrado Priami and Paola Quaglia. Beta binders for biological interactions. In *Proc. of Computational Methods in Systems Biology (CMSB'04)*, Lecture Notes in Bioinformatics. Springer, 2004.

[Pri96] Corrado Priami. Stochastic pi-calculus with general distributions. In *Proc. of the 4th Int. Workshop on Process Algebras and Performance Modelling (PAPM'96)*, 1996.

[PRSS01] Corrado Priami, Aviv Regev, Ehud Shapiro, and William Silvermann. Applications of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80(1):25–31, 2001.

[PV05] Catuscia Palamidessi and Frank D. Valencia. Recursion vs replication in process calculi: Expressiveness. *Bulletin of the European Association of Theoretical Computer Science*, 87:105–125, 2005.

[Reg03] Aviv Regev. *Computational Systems Biology: A Calculus for Biomolecular Knowledge*. PhD thesis, Tel Aviv University, 2003.

[Ric53] Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematics Society*, 74:358–366, 1953.

[RPS$^+$04] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. BioAmbients: An abstraction for biological compartments. *Theoretical Computer Science*, 325(1):141–167, September 2004.

[RS02] Aviv Regev and Ehud Shapiro. Cells as computation. *Nature*, 419(6905), September 2002.

[RSS01] Aviv Regev, William Silverman, and Ehud Shapiro. Representation and simulation of biochemical processes using the π-calculus process algebra. In *Proc. of Pacific Symposium on Biocomputing (PSB 2001)*, pages 459–470, 2001.

[Saï00] Hassen Saïdi. Model checking guided abstraction and analysis. In *SAS'00: Proc. of the 7th International Static Analysis Symposium*, volume 1824 of *Lecture Notes in Computer Science*, pages 377–396, London, UK, 2000. Springer-Verlag.

[Shi88] Olin Shivers. Control flow analysis in scheme. In *PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, pages 164–174, New York, NY, USA, 1988. ACM Press.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[Tay99] Paul Taylor. *Practical Foundations of Mathematics (Cambridge Studies in Advanced Mathematics)*. Cambridge University Press, May 1999.

[Tur36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[van05] Jan vanderGreef. Systems biology, connectivity and the future of medicine. In *IEE Proc. Systems Biology*, volume 152, pages 174–178, 2005.

[Ver07] Cristian Versari. A core calculus for a comparative analysis of bio-inspired calculi. In *Proc. of 16th European Symposium on Programming (ESOP'07)*, 2007. To Appear.

[Vil07a] Mariana Ruiz Villarreal. Cell illustration. Part of Wikimedia Commons, April 2007. Released into Public Domain.

[Vil07b] Mariana Ruiz Villarreal. Cell membrane illustration. Part of Wikimedia Commons, January 2007. Released into Public Domain.

[vMJ$^+$07] Jan vanderGreef, S. Martin, P. Juhasz, A. Adourian, T. Plasterer, E. R. Verheij, and R. N. McBurney. The art and practice of systems biology in medicine: Mapping patterns of relationships. *Journal of Proteome Research*, 2007.

[vSvdH04] Jan vanderGreef, Paul Stroobant, and Rob van der Heijden. The role of analytical sciences in medical systems biology. *Current Opinion in Chemical Biology*, 8:559–565, 2004.