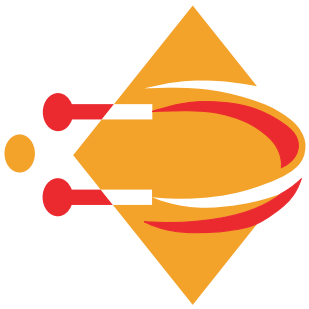
Bigraphs: Modelling, Simulation, and Type Systems

On Bigraphs for Ubiquitous Computing and
on Bigraphical Type Systems

Ebbe Elsborg



Dissertation
Successfully Defended on 16 March 2009
in Front of an External Evaluation Committee and
the Faculty of the IT University of Copenhagen
in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Programming, Logic, and Semantics group 24 March 2009

Abstract

We study how bigraphical reactive systems may be used for modelling and simulating — in a manner controlled by sorts and types — global ubiquitous computing. Ubiquitous computing was in the early 1990s envisioned by Mark Weiser to be the third wave of computing (after mainframes, and then personal computers), in which each person has many computers, receding into the background, at his or her disposal. Global ubiquitous computing has been identified internationally, indeed it is a UK Grand Challenge, as one of the most important challenges for computing in the 21st century.

In mathematical modelling of real-life systems, we aim to gain a deeper understanding of their behaviour by studying their essence. Concurrent and mobile systems are notoriously hard to understand in their entirety and thus to give guarantees about. The complexity increases when global ubiquitous computing is considered, especially as systems become larger. A key observation is the importance of understanding this new paradigm before it is realised by technological advances, because trustworthiness will be paramount, and theory is an important factor in achieving this. To this end, computer simulations complement well mathematical methods.

In this dissertation we focus on context-aware computing — in particular location-aware computing — which is at the core of ubiquitous computing. Our point of origin is the theory of Bigraphs by Milner and co-workers. Up until now it has been an open question whether Bigraphs is a suitable model for ubiquitous computing, for which we provide a partial answer. The relevant literature is surveyed and Bigraphs challenged by modelling and simulation of location systems. Plato-graphical models are developed as a more advanced modelling technique. Finally, exploiting Bigraphs as a meta-model not just for (process) calculi but for type systems, and to control models, we show how to develop inductive type systems for Bigraphs.

We thus expand the knowledge of what may be achieved with Bigraphs.

Preface

This dissertation evolved out of the efforts to evaluate bigraphical reactive systems as a model of global ubiquitous computing [Wei93, Wei91, CCK+05]; efforts undertaken by my colleagues in the Bigraphical Programming Languages (BPL) group at the IT University of Copenhagen (ITU) and I. Early on, in the spring of 2005, we realised that the concept of ubiquitous computing was not at all well-defined. It was, however, clear that the notion of location-awareness was important for the understanding of ubiquitous computing and this concept was somewhat more tangible; a (mobile) device can be "aware" of its location (or even its context in a broader sense) by receiving events from hardware sensors, which it uses to maintain a location model – a data structure – of the last known locations of the devices of interest. I set out to model location-aware systems using bigraphical reactive systems during the spring of 2005 and discovered that modelling location-aware systems using a single bigraphical reactive system was troublesome, because implementing location queries of location-aware applications on location models with reaction rules left me lacking certain control structures of traditional programming.

To escape from this predicament my adviser Lars Birkedal had the idea of lifting the level of abstraction by introducing the so-called 'Plato-graphical models', where three (concurrent) bigraphical reactive systems are used in modelling location systems and their environment; a model of the real world, a model of the model of the real world including a sensor component, and a model of the application running on a mobile device using the location information. This work was carried out in the summer and early autumn of 2005. Separating concerns turned out to be very useful in that the troublesome parts of the model could now be represented in a more structured programming language, MiniML, and then translated into a bigraphical reactive system by mapping terms to bigraphs and structural

operational semantics to reaction rules.

Having an idea of how to model even large systems, the BPL group set out to develop a tool to rewrite bigraphs thus implementing bigraphical reactive systems. Such a tool would enable us to experiment with the design and behaviour of (ubiquitous) systems through simulation, which would be a significant step toward our goal of evaluating Bigraphs as a model of global ubiquitous computing – bridging theory and practice. With Arne Glenstrup and Espen Højsgaard (and in the early phases also Troels C. Damgaard and Martin Elsman) as driving forces a prototype of the tool was released in 2007. Concurrently, I was working with my former co-adviser Henning Niss on the ambitious implementation of a realistic location system as a Plato-graphical model. Finally, in 2008, both the prototype of the BPL tool (see `http://www.itu.dk/research/pls/wiki/index.php/BPL_Tool`) and the model were ready for experimentation, and we were able to conduct (simple) simulations.

During the winter of 2007/2008 I spent four months at the University of Bologna completing my long-term term research stay abroad. Under Davide Sangiorgi's skillful guidance, I worked on a novel approach to type systems for (process) calculi. The idea wss, roughly, to push the problem of designing type systems and proving properties about them to the more abstract level of Bigraphs, as Bigraphs had already been established as an expressive meta-model for many process calculi. The main advantages of this approach are: a meta-model can describe several concrete calculi, therefore one can hope that a result for a meta-model can be transferred to all of these calculi; understanding type systems at the level of meta-models can help us to achieve a deeper understanding of the type systems themselves. This line of work is also relevant for modelling and simulation efforts as modelling with bigraphical reactive systems is somewhat like declarative programming, and static typing is indeed helpful when programming. By studying inductively defined type systems for Bigraphs one can directly recover traditional type systems and their guarantees for process calculi, and even obtain novel ones.

So, ultimo 2008, we have a bigraphical model of a symbolic location model, a prototype tool for normalisation, matching, and simulation of binding BRSs, and theories for typing and sorting to control models.

Papers

During my PhD studies I published four papers in peer-reviewed international conferences and journals, and one position paper at a workshop.

First, working in a different field of formal semantics I co-authored a conference paper on compositional specification and automatic analysis of commercial contracts, which was published at the 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA'04) [AEH+04]. Second, I co-authored a superseding journal article that appeared in a special issue on Leveraging Applications of Formal Methods in the International Journal on Software Tools for Technology Transfer (STTT) in 2006 [AEH+06]. I made a proportional contribution to both papers.

Third, I co-authored a paper on bigraphical models of context-aware systems, which was published at FoSSaCS'06 [BDE+06].

Then, I co-authored a position paper on bigraphical programming languages for pervasive computing, which was published at the 1st International Workshop on Combining Theory and Systems Building in Pervasive Computing (CTSB'06) [BBD+06], a Pervasive 2006 workshop.

Fourth, I co-authored a paper on inductive type systems for Bigraphs, which was published at the 4th Symposium on Trustworthy Global Computing (TGC'08) [EHS09].

The work on modelling and simulation of location-models is for the most part documented in technical reports and has yet to be polished and submitted for publication. The work on inductive type systems and their relation with predicate sortings is in progress, we have no firm results at this time of writing.

This dissertation subsumes and integrates the following papers, in order of appearance:

[Els06] Ebbe Elsborg. Bigraphical Location Models. Technical Report 94, IT University of Copenhagen, September 2006.

[BDE+06] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Hildebrandt, and Henning Niss. Bigraphical Models of Context-aware Systems. In Luca Aceto and Anna Ingólfsdóttir, editors, *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, volume 3921 of *Lecture Notes in Computer Science*, pages 187-201. Springer-Verlag, 2006.

[BDE+05] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Hildebrandt, and Henning Niss. Bigraphical Models of Context-aware Systems. Technical Report 74, IT University of Copenhagen, November 2005.

[EHS09] Ebbe Elsborg, Thomas Hildebrandt, and Davide Sangiorgi. Type Systems for Bigraphs. In Christos Kaklamanis and Flemming Niel-

son, editors, *Proceedings of the 4th International Symposium on Trustworthy Global Computing (TGC'08)*, *Lecture Notes in Computer Science*. Springer-Verlag, 2009. To appear.

[EHS08] Ebbe Elsborg, Thomas Hildebrandt, and Davide Sangiorgi. Type Systems for Bigraphs. Technical Report 110, IT University of Copenhagen, October 2008.

Chapters 2, 4, 5, 6, and 7 are slightly revised versions of the technical report [Els06] – the first section of Chapter 8 too. Chapter 3 subsumes and integrates [BDE$^+$06] and [BDE$^+$05]. Chapter 10 subsumes and integrates [EHS09] and [EHS08].

The following papers are omitted from this dissertation:

[AEH$^+$04] Jesper Andersen, Ebbe Elsborg, Fritz Henglein, Jakob Grue Simonsen, and Christian Stefansen. Compositional Specification of Commercial Contracts. *Preliminary Proceedings of the 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA'04)*, pages 103-110. University of Cyprus Report TR-2004-6, 2004.

[AEH$^+$06] Jesper Andersen, Ebbe Elsborg, Fritz Henglein, Jakob Grue Simonsen, and Christian Stefansen. Compositional Specification of Commercial Contracts. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):485-516, November 2006. Special Section on Leveraging Applications of Formal Methods.

[BBD$^+$06] Mikkel Bundgaard, Lars Birkedal, Søren Debois, Ebbe Elsborg, Arne J. Glenstrup, Thomas Hildebrandt, Troels C. Damgaard, Robin Milner, and Henning Niss. Bigraphical Programming Languages for Pervasive Computing. In Thomas Strang, Vinny Cahill, and Aaron Quigley, editors, *Pervasive 2006 Workshop Proceedings – The 1st International Workshop on Combining Theory and Systems Building in Pervasive Computing (CTSB'06)*, pages 653-658, 2006.

The first two are omitted because they fall within a different line or field of research; they are about formal semantics but have nothing to do with Bigraphs. The third paper omitted is a position paper.

The above mentioned works do not represent the totality of my work as a PhD student. To fulfill the requirements for the degree of doctor of philosophy I also undertook significant teaching responsibilities (approximately 500 hours of lecturing, project/thesis supervision, and exercise lessons), attended courses, seminars, summer schools, and conferences, gave talks, and peer-reviewed papers and articles.

This dissertation was successfully defended on 16 March 2009 in front of 1) an evaluation committee consisting of two external examinators — Marino Miculan of the University of Udine and Arne Skou of the University of Aalborg — with Jens Chr. Godskesen from the ITU as chairman, and 2) the faculty of the ITU, in partial fulfillment of the requirements for the degree of doctor of philosophy (*PhD* or *ph.d.*).

Acknowledgments

First and foremost I wish to thank my lovely wife Dagmar and our precious daughter Isabella for being the lights of my life. I also wish to thank my parents Vibeke and Erik for being there for me always, and my aunt Birgitte and uncle Kim for their moral support and for inviting us to dinner so often.

I would also like to thank the members of the BPL group at the ITU for constituting a stimulating research environment, and the following people in particular: Mikkel N. Bundgaard for being an outstanding office mate and for answering many of my questions; Søren Debois for enlightening discussions — primarily about Bigraphs — his unique sense of humour, and for being helpfully LaTeX savvy; Troels C. Damgaard for conversations about Bigraphs and life at large; my adviser Lars Birkedal for sound advice and ample patience during my PhD education; my former co-adviser Henning Niss for working with me and for his encouragement; and Thomas Hildebrandt for his inherent friendliness and our research collaborations.

I was delighted and thankful to spend four months — during the autumn and winter of 2007/2008 in Bologna — working with Davide Sangiorgi, who was a highly amiable host and collaborator. I also thank Davide's PhD student Jorge Andrés Pérez Parra for being so hospitable.

Happy thoughts go to my friends Jakob Lemvig, Jesper Sandvig Mariegaard, Jakob Grue Simonsen, and Philip Bille who through social interaction motivated me to complete my PhD degree.

Finally, I would like to thank the members of my evaluation committee Marino Miculan, Arne Skou, and Jens Chr. Godskesen for useful comments on the dissertation and a good experience during my defence.

The work in this dissertation was funded in part by the Danish Research Agency (grant no.: 2059-03-0031) and the IT University of Copenhagen (the LaCoMoCo/BPL project).

Contents

Abstract *page* iii
Preface v
Contents xi

I Preliminaries **1**

1 Introduction 3
1.1 Global ubiquitous computing 3
1.2 Theory versus engineering 6
1.3 Bigraphs 9
1.4 A model for global ubiquitous computing 16
1.5 A meta-model for type systems 20
1.6 A model for stating and proving properties of systems 22
1.7 Summary 22

II Modelling and Simulation **25**

2 Location Models 29
2.1 Introduction 29
2.2 Relationships, queries, and requirements 32
2.3 Classification of location models 36
2.4 A model of a reflective building 41
2.5 Concluding remarks 42

3 Bigraphical Models of Context-aware Systems 43
3.1 Introduction 44
3.2 Bigraphs and bigraphical reactive systems 45
3.3 Naive models of location-aware systems 48

3.4 Plato-graphical models of context-aware systems 50
3.5 Examples 53
3.6 Discussion 57
3.7 Conclusion and future work 59
3.8 Acknowledgments 60
3.A Encoding of "find all devices" 60
3.B Native queries 62
3.C Rigid control-sortings and RPOs 68

4 Encoding MiniML with References in Bigraphs 73
4.1 Purpose 73
4.2 Non-interference of closed links 74
4.3 Encoding references via closed links 77
4.4 Dynamic correspondence 91
4.5 Discussion 92

5 A Real-life Location Model 95
5.1 A reflective building 95
5.2 The model 96
5.3 Concluding remarks 120

6 Simulation of Location-aware Systems 123
6.1 An abstract location model 124
6.2 A pedagogical scenario 127
6.3 Simulation with the BPL tool 131
6.4 Case study: a tour guide 139

7 Related Work 141
7.1 Modelling 141
7.2 Context UNITY 143
7.3 Contextual Reactive Systems (CRSs) 149
7.4 A calculus for context-awareness (CAC) 153
7.5 A formal model for context-awareness (CONAWA) 156
7.6 Other approaches 159
7.7 Concluding remarks 160
7.8 Simulation 161

8 Future Work in Modelling and Simulation 163
8.1 Modelling 163
8.2 Simulation 169

9 Summary of Modelling and Simulation 171

III Type Systems **173**

10 Type Systems for Bigraphs 177
10.1 Introduction 178
10.2 Bigraphs 181
10.3 A bigraphical i/o-type system 187
10.4 Conclusion 194
10.A Full proofs 196

11 On Type Systems and Sortings for Bigraphs 215
11.1 Introduction to sortings 215
11.2 Inductive sortings 218
11.3 Summary 221

IV Conclusion **223**

12 Summary 225

13 Future work 229

Appendix 231
A.1 Background Bigraph definitions 231
A.2 Code 239
A.3 π-calculus 276

Bibliography 279

Part I

Preliminaries

1

Introduction

This dissertation is a contribution in meeting the UK Grand Challenge of Global Ubiquitous Computing. We study Bigraphs [JM04, Mil06a, Mil05a, JM03]: their theory and applications. We do so by initiating an evaluation of the theory of Bigraphs[1] as a model and simulation environment for global ubiquitous computing, and by exhibiting how one may *inductively* give static type systems for Bigraphs to better control the structure and behaviour of models.

Here is a brief outline of this introduction. First, we discuss the motivation, characteristics, and challenges of global ubiquitous computing. Second, we consider the gauge between theory and engineering (practice) that needs bridging. Then, having set the scene, we introduce Bigraphs as a theory within the field of concurrency and as a possible model for ubiquitous computing. Having the framework in place, we give the reader a teaser on how one can derive inductive type systems for process calculi using the fact that Bigraphs is a meta-model. Finally, we discuss how sorting can be used to control models and refine derived labelled transtition systems. All in all, the introduction aims at establishing motivation, setting the scene, and hinting at how our developments in this dissertation fit in.

1.1 Global ubiquitous computing

Ubiquitous computing (ubicomp) was envisioned by Mark Weiser [Wei93, Wei91] to be the third wave of computing (after mainframes, and then personal computers), in which each person has many computers, receding into the background, at his or her disposal. Ubiquitous computers will be

[1] We often write simply 'Bigraphs' instead of 'the theory of Bigraphs'. This term encompasses bigraphical reactive systems. Moreover, the term 'bigraph' refers to morphisms of a particular s-category.

able to interact and form ubiquitous computing systems, their aggregation being the *Global Ubiquitous Computer*.

Global ubiquitous computing (GUC) is one of the UK Grand Challenges of Computing Research[2]. The work in this dissertation falls within one such grand challenge, namely 'Ubiquitous Computing: Experience, Design and Science'[3] [KMS04, CCK+05]. These challenges propose to unite efforts of researchers to meet challenges that are envisioned to become essential in theory and practice within the upcoming decade. Often, theory trails practice and thus rarely has important impact on how problems are solved in practice. With regard to computers and computing systems this is emerging a huge problem because these are becoming increasingly complex – too complex for humans to completely understand in their whole.

1.1.1 Concurrency, distributivity, and mobility

Three factors that render contemporary computing systems particularly complicated are concurrency, distributivity, and mobility, aspects that are being studied both in theory and practice currently and have been so for a few decades. Concurrency is the branch of computer science that deals with concurrently executing programs or processes that may interact at any time, and it is this non-sequentiality that makes them powerful and complicated. Distributivity introduces interconnected locations where computation can take place, but these connections may fail at any time, introducing the need for complicated link failure protocols in computer networks. Mobility denotes the scenario where interconnected computing devices have a location, often in physical space, and where they can change location during computation, either objectively or subjectively, thus altering the network topology. All of these three, say, paradigms, introduce possibilities on their own and thus also complexity into systems that was not there in the times of sequential computing. Moreover, these paradigms may even be combined, yielding quite unwieldy systems.

1.1.2 Pervasive computing

These are the challenges that we face, but computing systems are envisioned to become *ubiquitous* or *pervasive* over the next decade or two [CCK+05]. This means that a burgeoning population of ubiquitous computers — some invisible to the naked eye — will be everywhere around us embedded in the

[2] `http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/`

[3] `http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/index.html`

fabric of our homes, work places, the public space, and even in our bodies. These computers may: interact with us and with each other to cooperate in solving tasks – sometimes even seamlessly; be very small and exist in dizzying numbers forming large complex computing systems, which demands scalability of design; be globally connected forming a truly large-scale system; and be "aware" of their context (physical or computational) or perhaps even "self-aware" thus exhibiting introspective behaviour. Imaginable is also for them to become self-organising and self-repairing. These ubiquitous computers will be "everywhere" doing "everything" so they should evidently be trustworthy. Hence, it is critical that we understand ubiquitous computers and their interaction properly before they become a reality. These, perhaps, dizzying possibilities of "tomorrow's" computing systems put high demands on their foundations indeed.

In this context, the term 'structure' refers to the ways entities interact and move among each other, as captured by structural theories of processes. In [CCK+05], the authors envision that no later than the year 2010 will we have a calculus or logic, which allows us to model systems such as a location-aware building. It is to be expected that models of real-life systems will need to capture not only *time* as a continuous variable, but also *continuous space*; according to [CCK+05] one approach is to use *hybrid automata* (modelling/representing both time and continuous space) governed by differential equations. Also, stochastic or probabilistic information will likely be needed to, e.g., faithfully model and simulate device movement in a sentient building, because sensors are not perfect. The sensor system usually approximates the location of devices by sampling measures from different (close by) sensors.

1.1.3 Computing in space

As if the previously described challenges were not enough, it furthermore makes perfect sense to consider both physical and virtual space, as argued in [Mil02]. "Computing in space" is about communication across space and actually considers the global computer as both a physical and a virtual entity. The term *infodynamics* is used to describe the fact that physical devices move in physical space but also in virtual space via their representations. *Infostatics* is the term used to state that software superposes virtual space upon physical space. The conclusion of [Mil02] is that joining the forces of software engineering and software theory is necessary to achieve success with the global computer.

1.1.4 Theory-driven development

Mathematical analysis of complex systems such as UCSs should be driven by experimental research because it seems impossible to foresee all the potential problems of this new computing paradigm. Real systems should be modelled and analysed, and hopefully theory can impact the way engineers build systems. A help in understanding UCSs could be a graphical representation and reconfiguration of, e.g., network topology. We would like to contribute here also by gradually formalising and reasoning about increasingly realistic systems; a first step is taken in this report. One could also imagine a family, or tower [Mil09, Mil06b], of models with consistency requirements between them, where each model aids in reasoning about a certain level of a UCS.

1.1.5 Science, toolkits, and theories

The required science, toolkits, and theories for global computing do not yet exist. Progress has been made, but we shall need much more supportive science to really influence engineering of the technologies and devices ensuring sufficiently correct and trustworthy behaviour. By announcing a challenge uniting researchers a decade or two ahead of the technological advancement, we can hope to understand the vision well enough to be able to influence how these systems are built when the technology is ready.

1.1.6 Goals

Essentially, the ideal goals of the Ubiquitous Computing grand challenge are: to define a set of design principles for global ubiquitous computing, and to develop science whose concepts, calculi, theories, and automated tools allow *predictive* analysis of global ubiquitous computing.

1.2 Theory versus engineering

Theory and engineering should be a combined effort to realise the potential of global ubiquitous computing [Mil02, Ter06]. This is a theoretical computer science dissertation so naturally our main interests lie within the theory perspective, thus let us delve a little further into that. Some central concerns that theories for ubiquitous computing must address are [CCK+05]: Structure, interaction, context-awareness, self-reconfiguration, information flow, and trustworthiness.

1.2.1 Structure and interaction

Structure and interaction have been two prime notions of process calculi over the last quarter-century. Three issues of particular importance have been placing, linking, and mobility.

Placing

Let us first consider placing. One way to structure entities, or processes, has been to use a hierarchy, e.g., a tree. A popular operator for concurrent calculi has been parallel composition as used in, e.g., π-calculus [Mil99, SW01], which gives rise to a flat process structure. Another popular process structure is an unordered tree as featured in the nesting of ambients in Mobile Ambients [CG00], and also in Distributed π-calculus (Dpi) [Hen08] where explicit locations holding processes are introduced to capture distributivity in the setting of message-passing. Thus, structurally, Dpi combines π-calculus and Mobile Ambients.

Linking

Linking describes the interconnection of computational nodes in a place structure. Usually, such links represent communication channels as, e.g., in π-calculus, where process interaction proceeds by passing messages over channels (links) that can be either base values such as integers or even links. Some process calculi allow process passing and are dubbed higher-order.

Mobility

The notion of mobility rests upon a notion of locality. Mobility then means the movement of a process from one location to another. One way to represent such movement is to define a process' location as its set of links, as in π-calculus. If links are passed over links, e.g. in π-calculus, then mobility is obtained because a location of a process is defined as the set of its links, and the reception of a new link will thus change the location.

Sometimes, the linking of a process defines its location, as in π-calculus, but there are other options. A process' location may define its linking, e.g. in the case of location-based services. Moreover, we may even consider placing and linking as two orthogonal structures. Thus, placing may affect linking and vice versa. Mobility certainly affects placing, but may also affect linking.

1.2.2 Context-awareness

Context-aware computing is a scenario where entities (e.g. mobile devices) are aware of their surrounding environment, i.e., adapt their behaviour depending on the context at hand [SAW94], interpreting 'context' to mean the situation in which the computation takes place [DA00]. Context-aware systems typically have a component that maintains a model of the current context, and such components are known as *context models* [HIR02].

The prime example of context is physical location, i.e., a device adapts its behaviour according to its perception of its physical location – a perception that it may accumulate from hardware sensors or by a wireless signal from a location system hardware infrastructure.

1.2.3 Information flow

As mentioned in [CCK+05] one can expect a merging of research on semi-structured data and process models to handle that movement of data and processes is becoming alike. With respect to information the need to query distributed data will likely be paramount. An example of work in this area is Reactive XML [HNO06, HNOW05], which is a bigraph-and-XML-based approach. Recently, modelling and verification of protocols for communication in mobile ad hoc networks (MANETs) [God08, GHK08, God07, GG06, God06, NH06, NH04] has become a lively research area, where issues of trust and resource access challenge formal models.

1.2.4 Trustworthiness

In essence, a system is trustworthy if we can guarantee that its behaviour lives up to some predefined specification. Such specifications, e.g. of concurrent, mobile, or even ubiquitous computing systems, can be difficult to make complete and error free. Moreover, it may be extremely hard to verify that an actual implementation lives up to the specification. Often, only an abstract or core part of a system can be proved trustworthy. We need computers and (semi-)automated tools to aid us, because models are becoming so extensive and complicated that a human can not comprehend them entirely.

Model checking is an approximately 25 year old successful branch [GV08] of research in computer science. A piece of software, the model checker, takes as input a mathematical model of a real-world system and a specification of desired properties, and as output it reports whether the model fulfills the properties. Often, the search space of models becomes

too large for a brute force approach and approximation techniques will have to apply. Approximations can yield 'false negatives', but never 'false positives'. This means that a property may hold for a model even though the model checker can not verify this, but it must never be the case that a property is reported to hold when, in fact, it does not. Probabilistic model checking is a more fine-grained technique that allows guarantees that are not 100 percent, and it is typically possible to dually relate guarantee percentages with model size. The gain is feasibility. The loss is precision.

There are other techniques for establishing trustworthiness of a system, but we do not discuss them here.

1.3 Bigraphs

Bigraphs is a theory, or model, in the field of concurrency theory. We study concurrently executing computer programs by identifying their essential characteristics and representing these as mathematical objects. The hope is then that when deepening our understanding of these objects we will also gain insights into the programs that can be represented by these objects.

In this section we will provide the reader with intuitions about Bigraphs, and try not to bog down the reader with formal definitions. The formal definitions are, of course, more precise, so we have included the most important ones in Appendix A.1, for the interested reader to find. They should not be necessary for this chapter, though, but later on things become more technical.

1.3.1 Concurrency theory

In concurrency theory we are particularly interested in process calculi or process algebra, where programs are represented by terms/processes. We study equivalences between processes in the hope of learning about equivalences of programs. Interesting equivalences should at least be congruence relations — placing equivalent processes into identical contexts should yield equivalent processes — because that allows us to replace one process (program) by another equivalent one as part of a larger process (program). In other words, the equivalence should be contextual. In fact, process calculi are usually compositional to allow such modularity.

The theory of Bigraphs is a meta-model aiming to encompass calculi for concurrency and mobility. Bigraphs is in direct continuation of Reactive Systems à la Milner and Leifer [LM00a], butr neither theory is an instance of the other. Specifically, Bigraphs is a graphical model of computation

in which both locality and connectivity are prominent. Seen as a model of these two important aspects of global computing Bigraphs takes up the challenge to base all distributed computation on a graphical structure, recognising the inherent topological nature of global computing. It was tailored directly to comprise two of the most fundamental and important process calculi; π-calculus and Mobile Ambients.

1.3.2 Reactions and transitions

In concurrency theory the dynamics of processes is often presented by means of reductions (rewriting rules) of the form

$$a \longrightarrow a'$$

where a and a' are agents (process terms). In process calculi we often refine this reduction relation into a labelled transition relation of form

$$a \xrightarrow{l} a'$$

where l expresses the interaction between agent a and its environment or context. Labelled transition systems (LTSs) conveniently support the definition of behavioural preorders and equivalences, such as traces, failures, and bisimilarity. LTSs are usually compositional but can be quite complicated – a typical example is the notorious difficulty of establishing that bisimulation equivalence on processes is a congruence relation. Reduction systems on the other hand define the meaning of a term to be its possible internal actions. An advantage is simplicity. A disadvantage is that they may not be informative enough, because they state nothing about the possible interactions between a process and its environment.

Extending a reduction semantics to an LTS is often achieved in an ad hoc manner for every calculus. Bigraphs offers to uniformly derive labels from a given set of reaction rules, automatically. Given a source calculus by a set of terms and a reduction semantics, one represents the terms by bigraphs and the reduction semantics by bigraphical reaction rules, and then one automatically derives an LTS on which there by construction exists a congruential behavioural equivalence, which respects contexts and ignores insignificant structural differences.

In this section we will gently introduce the reader to the world of Bigraphs. Then, we will take a closer look at the main aims of this theory.

1.3.3 Bigraphs for theory and applications

There are requirements on Bigraphs from applications and theory. With respect to applications, the long-term aim of Bigraphs is to provide a model of computation on a global scale, such as the Internet [JM04]. Not only should Bigraphs be a mathematical model describing existing systems, but should preferably also guide the specification, design, and programming of future such systems.

Moreover, Bigraphs should be general enough to provide for both high-level and low-level descriptions of systems, and we want to be able to describe how the low-level model realises the high-level model.

Finally, the theory should encompass existing theories or formalisms for concucrrency and mobility.

1.3.4 A bigraph is a graph

Bigraphs is based directly on the important and well-studied mathematical objects known as graphs. This merits a nice graphical or visual representation of bigraphs, which is indeed useful when modelling spatial systems. There also exists a complete algebraic theory [Mil05a, DB06] for (Pure and Binding) Bigraphs.

A bigraph is a graph. Actually, it consists of two graphs, namely a *place graph* describing the locality and nesting of computational nodes, and a *link graph* describing the inter-connectivity of said nodes. These two structures are overlayed, as they share the node set, to form a bigraph. The nesting is non-cyclic and the links are hyperlinks, i.e., a link may connect more than two nodes. In fact, the place graph is a forest of trees each defining a *region*, where nesting is represented by the parent-child relationship between nodes.

Such a bigraph can, e.g., represent a process or a physical location. Each node has a *control* associated with it that determines its type or kind. Each control has a fixed number of *ports* that may link it to names. If two ports link to the same name then they are linked together. Such a link can be private, i.e., invisible to the context, if it is closed (by "putting on top of it" a particular bigraph).

Because bigraphs are contexts they not only have names in the outer interface (that the context sees), they also have names in their inner interface (that *parameters* see).

The place graph is a forest of trees so a bigraph can be *wide*. It may occupy more than one hole (or *site*) in another bigraph, in which case it has an outer width greater than one. Bigraphs are multi-hole contexts.

1.3.5 Reaction rules

Bigraphs consist of nodes and links representing, e.g., interconnected computational entities such as processes. Naturally, we wish to be able to describe dynamic systems so a structured way of rewriting one bigraph to another is included; *reaction rules*. They rewrite ground bigraphs (or *agents*), i.e., bigraphs which are terms as opposed to contexts, to ground bigraphs. Reaction rules are parametric meaning that they are specified over contexts (bigraphs with holes). Roughly, a reaction rule is a pair of contexts[4]; a left-hand side (*redex*) and a right-hand side (*reactum*).

Without going into detail, for an agent b to be rewritten by a reaction rule (R,R') one must find a context C and some parameters p, p' such that the rule matches the agent $b = C \circ R \circ p$; we omit some details for now. The result of the reaction is $b' = C \circ R' \circ p'$. (p and p' are related.) So, reaction rules are roughly pairs of bigraphs. *Matching* of bigraphs with reaction rules has been studied to some extent in recent works [BDGM06, Dam08], as it is at the core of the BPL tool.

Reaction rules can be wide, they are so if the redex (and then also reactum) are. In fact, the redex and reactum must have the same outer width (and names).

1.3.6 Variations of Bigraphs

There exist several different variations of Bigraphs. Most notably, the Pure Bigraphs where locality and linking are orthogonal structures, i.e., "where you are does not affect who you can talk to". One can add lexical scopes on links in Binding Bigraphs, which then locates some names at nodes. This is useful for encoding calculi such as π-calculus and λ-calculus in Bigraphs, as these calculi use name binders heavily. It is also practical for modelling. There is a third main variant, which is known as Local Bigraphs. Local Bigraphs remind us of Binding Bigraphs, but in Local Bigraphs all names are located at regions or sites, as opposed to pure Bigraphs where all names are global (none are located). Furthermore, a name may reside at several locations, as opposed to binding Bigraphs, where local names are unique. This difference is important for modelling, and also has some technical implications. For modelling it turns out that this multiple locality gives us an additional control structure when programming with Bigraphs in that it in some cases allows us to use a wide reaction rule instead of a whole

[4] And an *instantiation*. More on this later.

collection of non-wide rules. This may prove significant with respect to scalability of models. We will touch upon this further in Chapter 4.

A variant of Bigraphs where edges (closed links) are also assigned controls has been proposed by Bundgaard and Sassone in [BS06] to naturally capture the type of new channels – channels created by the ν operator – in typing derivations of π-calculus processes in a bigraphical model.

In a note, Milner has suggested to loosen binding Bigraphs to allow 'outward' binding, i.e., to allow a binding port on a node to exercise lexical scope over a sibling nodes' names. It is argued that this is desired for modelling in reflective buildings, see also Chapter 5.

Another strand of research is that of Directed Bigraphs by Grohmann and Miculan [GM08b, GM08a, GM07b, GM07a]. They generalise link graphs to be directed meaning that links are uni-directional. With this change the Fusion calculus has a bigraphical model [GM07b, GM08a]. Lately, direction on ports (negative ports) has been introduced to govern the access to resources [GM08b], with the understanding that "resource request flow" inside link graphs goes from controls to edges (through names).

Finally, Stochastic Bigraphs [KMT08] have been developed to enhance the applicability of Bigraphs, in this case to computational, molecular biology. The expressiveness of the framework has been demonstrated through an example of membrane budding in a biological system. Another use of stochastic information is for dealing with quantitative aspects of ubicomp such as quality of service (QoS). The paper presents a stochastic semantics for BRSs; a reduction and a labelled stochastic semantics for Bigraphs are defined, and it is proved that the two semantics are consistent with each other. Support of stochastic phenomena is certainly important for future work on modelling of real-life systems, as discussed in Chapters 5 and 8.

1.3.7 Bigraphs and category theory

A bigraph is also a morphism in a particular kind of category, an *s-category* [Mil06a]. (For connoisseurs: an s-category is roughly a strict, symmetric monoidal pre-category with a finite set of node identifiers (the support) for each morphism. A pre-category is roughly a category where composition of morphisms with overlapping support is undefined. Formal definitions are in Appendix A.1.) The objects of s-categories are called *interfaces*, they determine which bigraphs may be composed vertically (one on top of the other) by categorical composition. S-categories possess tensor products allowing bigraphs to be composed horizontally, i.e., juxtaposed (put next to each other). Choosing the morphisms as representatives for bigraphs, as

opposed to the objects, has the advantage that a natural way of composing smaller bigraphs into larger ones appears, namely by categorical composition (and tensor product). The last thing that we need to know about these s-categories, for reading this dissertation, is that they have a notion of support, which allows us to keep track of the identity of nodes.

Bigraphs, or more precisely, BRSs, are an instance of a more general (non-graphical) categorical framework; Wide Reactive Systems (WRSs). Without going into detail we point out that the Relative Pushout (RPO) theory of Leifer and Milner is used to derive LTSs for WRSs, in a way that leads automatically to behavioural congruences. This is a main feature of Bigraphs; to uniformly derive (a suitably minimal set of) labels from a given set of reaction rules.

Sassone and Sobociński [SS03] generalise s-categories to G-categories (a variant of 2-categories). Then, they build G-reactive systems (GRSs) with G-RPOs on these [SS05b], achieving that bisimilarity on the derived transitions is congruential. Moreover, they prove that any reactive system on an s-category can be encoded faithfully in a GRS, preserving induced transitions [SS05a, SS03]. Alas, there is an inherent difference between BRSs and GRSs; BRSs are naturally modelled as output-linear cospans, but only input-linear cospan bi-categories over adhesive categories [LS05] have G-RPOs for sure [SS05b]. Thus, BRSs are not merely an instance of GRSs. Grohmann and Miculan's work on Directed Bigraphs aims to generalise both kinds of bigraphs with an RPO construction subsuming both Jensen-Milner's and Sassone-Sobociński's constructions [GM07a].

1.3.8 Aims of Bigraphs

Robin Milner and collaborators have developed Bigraphs with two principal aims (1) and (2) below, but we intersperse a third and a fourth one, (3) and (4) below.

1. A meta-model for (process) calculi deriving labelled transition systems with congruential behavioural equivalences.

2. A model for ubicomp.

3. A meta-model for (process) calculi deriving type systems.

4. A theory for stating and proving spatial and temporal properties of systems represented as BRSs using spatial-temporal logics, like the spatial logic BiLog [CMS05].

In this subsection we consider the first, traditional aim. Then, in the three following sections, 1.4, 1.5, and 1.6, we consider aims (2), (3), and (4), respectively.

1.3.9 Bigraphs as a meta-model

Bigraphs may be viewed as a unifying meta-model of calculi for concurrency and mobility, because Bigraphs can be instantiated both with respect to terms and semantics to encompass many concrete calculi. That is the challenge of process theory. The space of possible calculi is daunting; behavioural specification and equivalence, and different notions of locality are widely varying across calculi for mobility. And as the systems increase in complexity, e.g., considering the domain of web services, then the primitives of the calculi become manifold, complicated, and intrinsically different from each other. Bigraphs was developed to naturally extend two important, say, core classes of process calculi, namely π-calculus and Mobile Ambients.

Encoding calculi

The idea is that Bigraphs can model a whole class of calculi for mobility and concurrency so one may hope that results for Bigraphs can somehow be transferred to the calculi that have bigraphical models. This branch of research has investigated Bigraphs as a meta-model of calculi for concurrency and mobility by representing concrete calculi, defined by terms and a reduction semantics, as bigraphical reactive systems: Petri nets [Mil04b]; π-calculus [Jen07, JM04, JM03]; CCS [Mil06a]; Mobile Ambients [Jen07], Homer [BH06]; λ-calculus [Mil04c, Mil05b]; and Fusion Calculus [GM07b].

Deriving labels

Some of the derived behavioural equivalences coincide with known ones for the source calculi, however, others do not, as is discussed in the dissertations of Høgh Jensen [Jen07] and Debois [Deb08]. This is a problem because it is difficult to fix the bisimulation equivalence if it turns out not to be as desired or expected. It could be in conflict with your real-world intuition, e.g., if it distinguishes terms that in your intuition should be equal. Or, it could simply be intrinsically different from the equivalence that you already know and was trying to derive. For LTSs one can alter the labels to eliminate unintended interactions between a process and its environment. Alas, for

reactive systems, where the LTS is automatically derived, that option is non-existent. There are two other options, though. One is to alter the reaction semantics, but this would ruin their virtue of being simple enough to be self-evidently correct. The other option is to alter the compositional structure of terms and contexts, which is exactly the approach advocated in the work on bigraphical sortings, see Section 1.5.

Expressivity

As mentioned in Bundgaard's dissertation [Bun07], another aspect of Bigraphs as a meta-model is expressivity by encodings. Clearly, Bigraphs is at least as expressive as the calculi that are encodable in Bigraphs. Moreover, because reaction rules can be defined freely, we expect that Bigraphs is highly placed in expressiveness hierarchies. (For a discussion of espressiveness hierarchies and the criteria for building them, we recommend the paper [Gor08] by Gorla.) One may even obtain a deeper understanding of a calculus by considering its encoding or representation as a BRS.

In principle, Bigraphs could be used to compare calculi by encoding them into Bigraphs. To compare two calculi with different terms and semantics one would encode them into Bigraphs, e.g., using two different BRSs, and then call on a notion of equivalence between BRSs to provide the comparison. Alas, no such notion exists currently. We discuss this briefly in Chapter 3.

Neither derivation of labelled transition systems nor the expressivity aspect of Bigraphs is the main subject of this dissertation, however, so we move on to the second aim, which is at the very centre of our investigations.

1.4 A model for global ubiquitous computing

Pervasively in this dissertation we study the modelling power of Bigraphs, i.e., how suitable is the set of primitives for modelling real-world phenomena such as ubiquitous computing systems.

It can be extremely hard to predict every possible behaviour of large systems. This is a problem that computers can help us with, because they can check possibilities fast, exhaustively, and systematically. Even in the cases where a complete search is infeasible, computers can tell us something new about systems by *simulating* models of them.

One gain of mathematical modelling of real-world phenomena is to make concepts and relationships precise, and to allow rigorous reasoning about them. Simulation can help predict the behaviour of large systems and

it allows for experiments with the design of models/systems. To help us control the behaviour of our models we may introduce (static) type systems, just as one does for traditional programming languages.

The second aim of Bigraphs, mentioned above, is less tangible than the first one. The reason is that ubicomp is still a vision, an idea, not an existing computational paradigm with hardware and software already in place, nor is it a well-defined (mathematical) object.

1.4.1 Narrowing the problem domain

To obtain a more tangible problem to attack we narrow the domain of investigation. An important facet of ubiquitous computing that has received much attention in the research literature is context-awareness.

The most commonly exploited instance of context is physical location, as witnessed by the literature and the context-aware systems and toolkits that have been implemented [SLP04, BD05, Dom01, SC02]. Location-aware applications acquire information from sensors, which can happen in a uniform way through a *location model* [BD05] that interprets sensor information to maintain a model of the current locations (positions) of, e.g., mobile devices. We delve into location models in Chapter 2. Location models are concrete enough for us to study and they are an essential part of location-aware systems.

1.4.2 Sentient computing

A foothill project 'Analysing Movement in a Sentient Environment'[5], under the GC in question, takes our train of thought of location-aware computing to a concrete scenario. *Sentient computing* [ACH+01, Hop00] is about how software applications can be made more responsive and useful by observing the physical world and reacting to it. Sensors (hardware) collect context (e.g. location) information of physical objects such as mobile devices and even other (mobile) sensors. We use the words sentient and context-aware interchangeably.

The foothill project aims to arrive at a conceptual framework in which to express a variety of rules of motion and interconnection, allowing context-aware systems to be programmed conveniently, simulated, and analysed rigorously. We use the term 'system' broadly to mean a group of independent but interrelated elements comprising a unified whole. We will detail this in Chapter 2.

[5] www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/fp-movement.html

The framework could consist of a calculus and a derived programming language along with a programming methodology so that the language may be used and evaluated by people whose primary interest is in applications. The ultimate goal of this foothill project is to unify theory and practice in sentient computing. A step toward this goal could be to model and program a sentient or "reflective" building, where sensors continually transmit data to a monitor that maintains a data structure of the locations of physical objects. A more advanced task could be to also model mobile virtual objects such as mobile code moving from one software domain to another.

1.4.3 Our approach

Our strategy toward meeting the challenge posed by this foothill project is to take an experimental approach by comprehensive modelling of a concrete, realistic system.

First, in Chapter 2 we study the literature on location models to gain understanding of the models we wish to model bigraphically.

Then, to test whether Bigraphs are suited for direct modelling of context-aware systems we pick some queries on location-aware systems to model. Such queries are concrete, and it is clear when a query has been implemented faithfully. If we can successfully handle the important facet of (physical) location in pervasive computing then we can reasonably hope to extend the work and techniques to the more advanced setting with several facets.

We find that direct modelling of a whole location system in a single BRS is inconvenient. There are two reasons for this. First, these queries are difficult to implement by bigraphical reaction rules only. Operationally, the queries involve inspection of the location model, which is essentially a tree traversal. It turns out, perhaps surprisingly, that we lack control structures to implement such an apparently easy recursive procedure. Second, modelling both the model and the queries on it in the same BRS results in conceptual problems, because the location system consists of several parts that evolve concurrently, and to some extent, independently. As an example, we point out that an executing query will change the state of the location model by use of auxiliary controls that manage the traversal of the tree structure. This is unnatural. We return to both of these points in Chapter 3.

To tackle these issues we develop a more advanced modelling method; Plato-graphical models. The idea is to split the model into four parts; a representation of the real world **C**, a sensor component **S** that observes **C**, a location model part **L** which is the model's representation of **C** that is main-

tained by information on changes in **C** mediated by **S**, and an application part **A** representing a location-aware application. **S** and **L** are encompassed by a single BRS **P**; the 'proxy'. To test this setup we model a context-aware printing example from the literature.

Further, we use our gained knowledge of location models to formulate a representative one, and model it as a Plato-graphical model, in Chapter 5. In doing this, we define an encoding of a MiniML-like calculus into Bigraphs, in Chapter 4, and analyse this encoding. Separating concerns in Plato-graphical models allows us to express parts of the model in another formalism than Bigraphs, if we can encode this formalism as a BRS. Concretely, we represent the operationally more involved parts **L** and **A** in a (small) functional language, MiniML, and then we translate these components into BRSs.

Having a model is useful. The next step is to simulate evolution of this model as this can help us experiment with the design of the system. A goal is to uncover unforeseen and unwanted behaviour, which is a notorious problem in concurrent systems, and in particular for large systems. We begin this branch of research in Chapter 6, where we go through two scenarios of an abstract version of the location system.

1.4.4 Modelling and simulation summary

Our point of origin is the theory of Bigraphs [JM04, Mil05a, JM03]. A principal aim of BRSs is to model ubiquitous systems. In this dissertation we begin evaluation of this aim. We find that:

1. It is awkward to model context-aware (location-aware) systems directly in Bigraphs.

2. This awkwardness can be alleviated by using *Plato-graphical* models.

3. Location models can be modelled as Plato-graphical models using a bigraphical encoding of a MiniML-like calculus with references.

4. Bigraphical models, in this case a Plato-graphical model, can be simulated using a prototype of the BPL tool, allowing for experimentation with the design of models and to some extent systems.

The BPL tool is still under development in the BPL group, in particular by Arne Glenstrup and Espen Højsgaard.

1.5 A meta-model for type systems

This section outlines two approaches to types for Bigraphs; inductive type systems and sortings.

1.5.1 Inductive type systems

Briefly, our idea is to design type systems for Bigraphs, prove properties of them, and then transfer them along with their properties to encodable source calculi that display a tight dynamic correspondence with their bigraphical models.

Obviously, the larger the class of encodable calculi is for a given BRS, the more work can be saved or, said in another way, the more results we obtain "for free". Likewise, if many type systems can be defined on top of the same BRS then we save more work.

In a little more detail, our idea is to define *inductive* type systems à la type systems for π-calculi, but on the term language of Bigraphs, i.e., the complete and sound term language generated from the small set of *elementary* bigraphs and the two operations of tensor product and categorical composition. This is a novel idea. It is inherently different from that of bigraphical *sorting* in that sortings functors are hardly ever inductively defined on the structure of bigraphs [Deb08]. Possible advantages of developing (inductive) bigraphical type systems include: a deeper understanding of a type system itself and its properties; transfer of the type systems to the concrete family of calculi that the BRS models; and the possibility of modularly adapting the type systems to extensions of the BRS (with new controls).

1.5.2 Sorting

The idea of sorting or typing bigraphs is not new. Traditionally, sorting has been the preferred term for typing names in process calculi. In Milner's book on the π-calculus [Mil99] a sort is assigned to every name to ensure that when a process receives a name it uses it properly. To this end a sorting is a partial function from a set Σ of sorts to Σ^* (the set of all sorts over Σ). Notice that sorting can be impossible for a given system, but if it is constructable then it is preserved by structural congruence and reaction (Prop. 11.5 of [Mil99]). Milner uses the term 'sort' to classify names, because he has reserved the term 'type' to classify processes. We remark that the notions of 'abstraction' and 'concretion' in Bigraphs also stem from π-calculus notions, see [Mil91].

In the setting of Bigraphs, sorting is essentially a mechanism to outlaw some unwanted bigraphs from the underlying s-category. Roughly, one specifies a predicate that stipulates, in terms of nodes and links, which bigraphs we like. It is a mechanism for altering the compositional structure of bigraphs, in such a way that also the altered category has sufficient structure for bisimulation to be a congruence relation on the derived LTS. A sorting merely restricts which bigraphs can go into which, and maybe outlaws certain bigraphs. It does *not* change the structure of the individual bigraph, but it does enrich the interfaces (objects) of the bigraphs; a sort component is added. Bigraphs with the same sorts in their otherwise common object can be composed. We will see a non-trivial example of sorting in Chapter 3.

Traditionally, such predicates were in plain English [Mil06a]. Debois, in his dissertation [Deb08], takes the functors that the predicates give rise to as the very definition of sorting, thus paving the way for an elaborate formal theory of these sorting functors. In both cases there are some restrictions on sortings, which secure that the bigraphs that are cut out of the category are none of those needed to construct RPOs, because if they were, the automatic derivation of the LTS with congruential behavioural equivalences would break down.

In works by Milner, Leifer, and Høgh Jensen, respectively, some sufficient conditions for "safe sortings" have been established. The conditions were not necessary though, one can find sortings that are "safe" without them satisfying the conditions. In the work by Debois and collaborators the predicates are required to be decomposable, which means that

$$P(f \circ g) \implies P(f) \,\&\, P(g)$$

for a predicate P and bigraphs f and g. In fact, given such a predicate their technical machinery will generate a safe sorting for you, but the category will have other objects than before, usually also more objects.

Bigraphs encompass both links à la π-calculus, but also nested nodes à la Mobile Ambients. Thus, both sortings for links and nodes have been proposed to aid bigraphical modelling efforts of such calculi. A prominent link sorting is the one for modelling Petri Nets in Bigraphs [LM06]. Another one implements subtyping on channels for a bigraphical model of polyadic π-calculus [BS06]. Høgh Jensen uses several place sortings in his dissertation [Jen07] to model increasingly complicated versions of π-calculus and Mobile Ambients. For π-calculus a place sorting is used to model summation, whereas for Ambients a place sorting is used to represent the difference between systems/networks and processes.

With respect to modelling, sorting actually gives one another tool. It enables the modeller to remove some unwanted bigraphs from the system, which effectively corresponds to the *inhibitors* of Braione and Picco's Contextual Reactive Systems [BP04, Bra03] and, in some cases, also the *negative application conditions* of Algebraic Graph Transformation [EEPT06]. See Chapter 7.

In Part III we explore inductive type systems for Bigraphs and relate this novel approach to the known approach of sorting. We find that even though the two approaches are very different there is a way to recover inductive type systems by formulating them as predicates and then using the theory of Debois and collaborators to generate appropriate sortings. In other words, we begin to bridge these two approaches in Chapter 11.

1.6 A model for stating and proving properties of systems

As will become evident, Bigraphs is a suitable formalism for modelling mobile and concurrent systems. To verify that models are well-behaved we may define a specification of the desired behaviour to check against. Such a specification could be formulated using a spatial logic such as BiLog and checked using a tool. One could construct a tool to work on the bigraphical term language of the BPL project, or translate the term language into a format on which the well-established model-checkers can work on, which would have the advantage that model-checkers for many different calculi exist; probabilistic, timed and so forth. Extending BiLog to handle binding and temporal information seems like important steps in this direction.

1.7 Summary

This concludes the overview of ubicomp, Bigraphs, modelling, simulation, and type systems and sortings. We have also touched upon some of the developments that this dissertation holds, and on related work.

This dissertation consists of four parts, where Parts II and III are the main ones. In Part II we treat modelling and simulation of context-aware systems, whereas in Part III we study inductive type systems for Bigraphs and their relation to sortings. Here is a brief outline of this dissertation.

- Part I: We introduced the UK Grand Challenge of Global Ubiquitous Computing along with a sketch of our contributions in modelling and simulation to meet this challenge. Furthermore, we provided an introduction to Bigraphs and type systems/sortings as this is our

point of origin. The introduction was interspersed with literature discussions.

- Part II: We survey the literature on location models and synthesise it to pinpoint general characteristics of location-aware systems, in Chapter 2. Upon that knowledge we develop Plato-graphical models for modelling context-aware systems in Bigraphs, in Chapter 3. In Chapter 4, we encode MiniML with references into Bigraphs and use this to develop a Plato-graphical model of a general location-aware system in Chapter 5. We "close the circle" by reporting on performed simulation of this system, in Chapter 6. Then, we discuss related and future work in Chapters 7 and 8, respectively. Finally, we summarise very briefly in Chapter 9.

- Part III: We begin this part with the development of inductive type systems for process calculi, in Chapter 10. As proof of concept we present a model of a core π-calculus as a bigraphical reactive system, develop an i/o-type system on elementary bigraph terms and their operators, prove crucial properties of it, derive a type system for the source calculus, and transfer these results back to the typed source calculus. Then, in Chapter 11, we sketch work-in-progress on how to develop inductive type systems for Bigraphs. In Chapter 11 we study the relationship between bigraphical inductive type systems and sortings.

- Part IV: We conclude and point out the most promising directions for future work.

Dear reader, I hope that You will enjoy reading this dissertation.

Part II

Modelling and Simulation

"That was to me the challenge: picking communicational primitives which could be understood in systems at a reasonably high level as well as in the way these systems are implemented at a very low level...But still, the emphasis ought to be on modelling what happens in real systems, whether they are human-made systems like operating systems, or whether they exist already...But as we move towards mobility, understanding systems that move about globally, you need to commit yourself to a richer set of semantic primitives. I think we are in a terrific tension between (a) finding a small set of primitives and (b) modelling the real world accurately."

– Robin Milner, [Ber03]

"Do not quench your inspiration and your imagination; do not become the slave of your model."

– Vincent Van Gogh[6]

"...I want to extend the computer and information systems to observe the real world and automatically modify the systems' behaviour to suit the prevailing conditions. Such "sentient" computing systems will play a key part in ensuring the sustainability of our planet."

– Andy Hopper[7]

"The problem of verifying was a problem of simulation or data representation, and I realised how big a problem that was going to be.

In fact, out of that I got interested in simulation, which I did a bit of work on."

– Robin Milner, [Ber03]

[6]Dutch artist, 1853-1890.

[7]`http://www.cl.cam.ac.uk/Research/DTG/~ah12/aims-research.html`

2

Location Models

2.1 Introduction

It is well agreed upon that *location* is an important context-parameter [Sch95, Leo98], but not the only one [SBG99] in context-aware computing, and that context-aware computing will become increasingly important in the years to come. This chapter serves as general background knowledge for Chapter 3 on bigraphical models of context-aware systems, and as basis for modelling a location model in Chapter 5.

2.1.1 Context-awareness

Location is the most important piece of context information. Therefore we concentrate on this in our work. If we can capture location then it should also be possible to extend our techniques to other facets of context-awareness such as battery power, ligthing, noise, temperature, user behaviour, and virtual context information such as libraries and concurrently running processes. We discuss the extension of our work from location-awareness to context-awareness in Chapter 8.

2.1.2 Location systems

First, we need some terminology.

Located-objects

As mentioned earlier, there are (at least) two ways to think about location; namely physical and virtual. In the present discussion we think of physical location, i.e., the location of objects in the physical world. Typically, it is

the location of real-world entities such as mobile devices (e.g. mobile phones) that is interesting for location-aware applications (which we explain shortly). Following [Leo98, ST94] we use the term *located-object* to refer to a mobile object whose physical location can be tracked.

The overall location system model

In this chapter we present a digest of the research literature on *location models*. We say that a location model is constituted by (1) representations of static and mobile real-world objects, (2) spatial relationships between these objects, (3) a collection of rules that model object movement, and (4) a collection of location information queries on the model.

Location models are essential parts of *location systems* (see [HB01] for a survey on location systems) because they provide a uniform way for applications to obtain location information, which facilitates rapid development. Before delving into location systems we need to address how information about location is presented in different formats.

Geometric coordinates, as used by the Global Positioning System (GPS), refer to a point or geometric figure in a multi-dimensional space.

Symbolic coordinates are names and can refer to cell-IDs in cellular networks such as the Global System for Mobile communications (GSM) or Wireless Local Area Networks (WLANs), or to radio frequency tags (RFIDs). The distinction between these two coordinate types is fundamental and we will return to it shortly. For now, please consider Figure 2.1, which depicts the overall system model. We explain Figure 2.1 in a top-down fashion.

A *location-aware application* (see [Leo98] for examples) queries a location model for location information. By location-aware we mean "the ability to adapt behaviour to the physical locations of users, resources, and processes" [Leo98]. In some cases the application can update the location model; we say that this is *actuation*. The different kinds of queries and actuators imply demands on the internal structure and organisation of the location model.

The location model maintains a representation of the state of the physical world by receiving events about updated position information on mobile objects from a *positioning system* (see [HB01] for an overview of positioning systems).

"A positioning system allows a mobile object or tracking system to issue a position update with a coordinate identifying a location to the location model." [BD05]. In [Leo98] it is stated that a positioning system measures the location of the querying located-object (e.g. vehicle navigation systems), whereas a tracking system measures the location of *other* located-objects

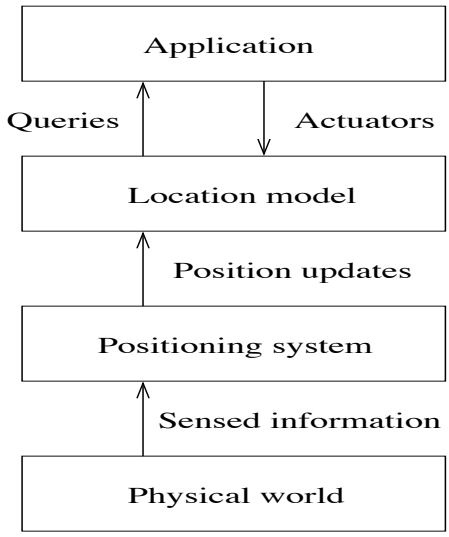


Figure 2.1: Overall location system model.

(e.g. the Active Badge system [WHFG92]). We do not wish to distinguish between tracking and positioning because we need objects to enquire about both their own and other objects' locations. If different positioning systems are in play then there is a need for *sensor fusion*, but that is out of the scope of this dissertation so we refer to [HBB02] for a discussion of a layered model for location in ubiquitous computing.

In the preceding explanation we have used the terms 'location' and 'position'. Following [HB01] we distinguish between *physical position* and *symbolic location*; a physical position is specified by a geometric coordinate, whereas a symbolic location is specified by a symbolic coordinate. The positioning system generates location information events on the basis of what its (hardware) *sensors* sense in the physical world. The sensors track the movement of located-objects, and the sensed information is delivered to the location model – usually via events that, in real systems, are time-stamped. The physical world is the world we live in, which is narrowed according to the geographical location of interest, e.g., a sentient building as in our case.

We sometimes wish to speak of a geographical point or area without being specific as to whether we consider it from a geometric or symbolic point of view. We overload the term "location" for this purpose, but usually in this dissertation we speak of symbolic locations.

Focus

We focus on a conceptual classification of the models. We do not discuss specific location-aware applications, positioning systems or sensor technology any further, except for a few comments later on.

2.2 Relationships, queries, and requirements

As mentioned earlier, location-aware applications query a location model. We intend to identify types of common queries, and mention which demands they list for the underlying location model. This requires us to first study coordinates and spatial relationships between locations. We do not consider actuation in this chapter, but we do return to it briefly during this dissertation and in Chapter 9. We proceed to explain location models and their basis.

2.2.1 Basic properties of coordinates

We follow the definitions of [BD05]. A coordinate is an identifier specifying the physical position of an object with respect to a given *coordinate system*, or the symbolic location by a *name* (e.g. a cell-ID). A coordinate system is a set of coordinates. As mentioned, there are essentially two different classes of coordinates, namely geometric and symbolic. We discuss each class of coordinates in turn.

Geometric coordinates

Geometric coordinates refer to a point or geometric figure in a multi-dimensional space and can be *global* or *local*. Geometric coordinates naturally support *calculation* of physical distance and containment relationships between positions (which are described by one ore more coordinates), to which we return shortly. Through calculation, geometric coordinates also support the following operations: Area overlap, areas touching, and area containment.

GPS is an example of a system using global geometric coordinates, where coordinates are triples of longitude, latitude, and elevation above main sea level. Many applications use GPS – e.g. navigation systems in cars. An example of a system using local geometric coordinates is the Active Bat system [ACH+01], which is a high-resolution indoor positioning system providing three-dimensional coordinates with respect to a local Cartesian

coordinate system. In other words the physical space is defined by a coordinate system, positions are identified by coordinate tuples, and the location model is geometric, i.e., identifies positions by geometric coordinates.

Symbolic coordinates

Symbolic coordinates are *names* that refer to locations, e.g. a room, a cell ID, or an IR identifier of a sensor. A reason for having symbolic coordinates is that they are "human-readable" – it is often more useful to know that a person is in a particular cell (e.g. a room), than at some given (set of) coordinate(s). Given only symbolic coordinates, it is *not* possible to *calculate* distances via a distance function, but the distance and containment relationships on locations (to be explained shortly) must be represented explicitly in the location model. Nevertheless, a symbolic notion of *nearness* (or *proximity*) can be supported, i.e., a located-object is close to another located-object or location. We return to this below.

The Active Badge system [WHFG92] provides symbolic identifiers (coordinates) for locations via fixed IR sensors registering users' badges that transmit a unique identifier. In other words the location space is defined by the placement of fixed sensors, a location is defined by the symbolic name of the sensor, and the location model is symbolic. Another application that uses symbolic coordinates is the Active Office system [WJH97] where locations are denoted 'building', 'floor', 'room' and so forth.

2.2.2 *Relationships of locations and located-objects*

In the literature five relationships pertaining to locations and located-objects are emphasised as having practical importance; *Contains* (inclusion), *connected-to*, *near* (proximity), *range*, and *distance*. These spatial relationships between locations are relevant for queries and topologies of location models. We briefly discuss each one in turn.

Contains Indicates whether a location is completely included in another. This relation is supported naturally in models with a hierarchical location structure such as trees and lattices. As mentioned, it can be calculated in geometric models. As an example: A building can contain a room, but hardly vice versa.

Connected-to Refers to some linking between locations. This relationship is often captured by introducing a graph-based location structure, as it

can not be calculated or derived. Examples: Two mobile devices can be connected, e.g., via Bluetooth. Two rooms can be connected by a door.

Distance The distance relationship is defined on spatial objects and is usually expressed as a natural or real number. It can be calculated in geometric models, but needs to be explicit in symbolic models. As an example: Two mobile devices can be positioned ten meters from each other, but in different rooms. This raises the question of how to calculate distance; by following a path via the connected-to relationship, or as the Euclidean distance. We return to this question in Chapter 5.

Near For a located-object l to be near another, a notion of distance function is required. The near-relationship could contain the n located-objects closest to the position of l. This relationship can be calculated in geometric models, but must be explicitly represented in symbolic models. It can be seen as a specialisation of the distance relationship. As an example: A user of a mobile device may want to find the nearest printer.

Range The range relationship has the located-objects within a certain geographic area of the located-object in question. To support this query located-object positions must be known and the contains relationship modelled, i.e., it has to be defined whether a coordinate lies within a spatial area. As an example: The sending of messages to receivers in a certain geographic area, e.g., a room on (contained in) the fourth floor (contained) in a building.

2.2.3 Queries

In [BD05] four different query types, which location models should support, are identified. We present and explain them for future reference. When explaining the queries we refer to the relationships.

Position queries: Determination of the position or location of a located-object like a user's mobile device, or a static object like a room. A position is defined by local or global coordinates. It would, e.g., be relevant with a local coordinate system for a moving train so that a traveller can be located in a compartment instead of his or her position on the ground [BD05]. To compare positions from local coordinate systems, mappings to a common global coordinate system must be defined.

Nearest neighbour queries: A search for the located-object or location closest to a certain position, e.g., a printer. Besides known located-object positions a distance function is needed to support this query. This function should output the physical distance when supplied with two coordinate tuples.

Navigation: Finding paths between locations. There is a need to model the topological connected-to relationship, which describes interconnections between neighbouring locations. This can, e.g., be used to find the shortest or fastest path, or a path for a person in a wheel chair. One could imagine adding weights to links for this purpose.

Range queries: Search for all located-objects within a certain geographic area. This can, e.g., be used to send messages to receivers in a certain geographic area as in Geocast protocols [DR03]. To this end, the model needs to be able to determine the positions of the located-objects and also the topological contains relationship, i.e., whether a coordinate lies within a spatial area. Containment is supported implicitly for geometric coordinates, but must be specified explicitly for symbolic coordinates.

These are the query types we will consider supporting in our bigraphical location model in Chapter 5. We should mention that [BD05] also has a requirement stating that all information of the location model can be visualised, but we do not consider that as a query as such.

2.2.4 Location model requirements

Having described these queries the following model requirements, not all of which need to be fulfilled at the same time, are derived in [BD05]. The requirements are on general-purpose models that wish to support all four query types, and so a model for a specific purpose need not necessarily fulfill these requirements to be of use. We believe that our preceding treatment justifies these requirements without further comments.

Object positions: Need geometric and symbolic coordinates to support a wide range of applications that have been implemented. Can do with either geometric or symbolic in some cases. Multiple local and global coordinate reference systems are desirable. This supports the position queries.

Distance function: Distances between spatial objects; Euclidean and desirably over paths. This is required for the nearest neighbour and geometric range queries. We argue that range can also be supported by location containment and can thus make sense in symbolic models also.

Topological relations: Contains and connected-to. These are needed to support range and navigation queries, respectively.

Orientation: Horizontal and vertical orientation is required for some applications, e.g., to determine which situation a person is in.

According to [Leo98] *co-location* is another interesting relative relationship. We consider this to be a range query where the range is exactly ones own location.

[BD05] argues that *minimal modelling effort* should be considered when constructing a location model, i.e., with respect to *accuracy* (creation and updating of the model, dynamics), *level of detail* (granularity of locations), and *scope* (the area covered; a building, a room, a country). We agree, and return to this in Chapter 5.

2.3 Classification of location models

When classifying location models we need to have a clear terminology. Unfortunately, there is no clear consensus in the research literature regarding the terminology of location model types. We proceed by synthesising the terminology of the literature. Some terms used are: Geometric, physical, symbolic, geographical, semantic, metric, topological, and Cartesian [HB01, Rot03, Pra00, BS01, BD05, DRD+00, BZD02, CK00]. We believe that these terms, in essence, cover two different types of location models; symbolic and geometric. We group the terms as follows:

Symbolic includes geographical, semantic and topological.

Geometric includes physical, metric, and Cartesian.

Hybrid models are combinations of symbolic and geometric models.

We continue by giving explanations of geometric and symbolic location models following [Leo98], along with brief justifications of our grouping. We begin with the geometric models and continue with the symbolic models.

2.3.1 Geometric location models

Geometric location models define the physical space by one or more multidimensional reference coordinate systems. Both positions and located-objects are represented as points, areas, or volumes within these coordinate systems. This supports calculation of the relationships distance (which may not be accurate) and containment between positions, and therefore also allows for calculation of area overlap, and whether areas touch. The connected-to relationship is however not inherent. A geometric location model is said to be *unified* if it has multiple coordinate systems, otherwise *simple*. Often, *uncertainty areas* are used to capture the imprecision of sensors (see e.g. [SBG99, HHS+02, HB01]) when positioning located-objects. Some geometric systems use *global* coordinates, e.g., referring to the position on the Earth's surface like in GPS, while other systems (e.g. Active Bat [ACH+01, HHS+02]) use *local* coordinates referring to a (smaller) Cartesian coordinate system with another point of origin (which is typical for indoor positioning systems). Geometric models use *absolute* positions, i.e., located-objects and locations are positioned with reference to some common point of origin (within each coordinate system), and *not* relative to each other.

Physical, metric, and Cartesian models

Physical, metric, and Cartesian denote exactly the same type of model as geometric.

2.3.2 Symbolic location models

In symbolic location models locations and located-objects are referred to by symbols or *names* such as "Room 4C.16" or "Linus Torvald's laptop". Symbolic models use *relative* locations meaning that each located-object has its own frame of reference because there are no underlying absolute positions. Locations can be organised in different structures to support different queries. We review three different approaches: Set-based models, graph-based models, and hybrid models combining the two approaches.

Set-based models

Locations can be modelled (naturally) as sets of located-objects which are represented by symbolic coordinates. A located-object is a member of a location whenever it is physically within the associated area or volume. Using sets, overlap of locations L_1 and L_2 is represented by set intersection

$L_1 \cap L_2 \neq \emptyset$, and thus also the containment relationship, if $L_1 \cap L_2 = L_1$ then L_2 contains L_1. This supports range queries by subset construction. It should also be possible to test for equivalence on locations. The support for queries related to spatial distances is naturally limited, but a notion of qualitative distance on symbolic coordinates can be modelled via set membership tests, we refer to [BD05] for the details. We mention two example systems; Guide [CDMF00] and Active Badge [WHFG92].

Cell models This is the most basic and flexible set-based model, and thus named a *simple symbolic* model. In this model the location space is described by cells so cells are the symbolic locations. A cell is a well-defined geographical area, e.g., a room. Cells can overlap, and need not cover the whole space. This is realistic with respect to sensor systems. There is typically no containment relationship in cell models. An example system is GPS, where the cell's area is a circle defined by the sighting coordinates and the accuracy margins [Leo98].

Zone models A zone model is a cell model with exclusive membership, i.e., non-overlapping locations. In a cell model cells may overlap. These overlaps are named zones. Each zone is part of one or more cells. Now, zones are used as symbolic locations. Imposing the constraint that locations must be non-overlapping yields an *exclusive symbolic* model. A single zone space can accommodate an arbitrary number of cells, which is useful if several sensor systems are in play. Since zones do not overlap, a located-object can be in at most one zone at a time. As noted in [Leo98] the movements of one located-object can be modelled by a single finite-state machine making the zone space a natural framework for persistent tracking and movement prediction. Imposing hierarchical locations via a partial order on a zone model yields a location tree structure. An example is found in [HHS+02] where a *quadtree* (a tree where all nodes have four children) is used. (Such structures support multi-resolution and thus scalability of design.)

Domain models A domain model is a zone model where locations (zones) have been partially ordered in a virtual hierarchy of domains. The aim is to enable multi-resolution tracking. A zone is a member of at most one domain. Domains are partially ordered, by the contains relationship, and can overlap. Thus, it is now possible to relate some zones to 'building B', which is part of 'Campus S' and also part of 'The computer science department', for example. If 'A' is a member of 'B' then it is also a member of the ancestors of 'B' in the domain ordering. Changes in domain membership

should propagate through the model. Multi-resolution refers to the ability to, e.g., say that a located-object is situated in room '4C16" or in 'ITU' (where '4C16' is a member of 'ITU'). See [DR03] for an example. If a lattice is imposed as the ordering, then a simple notion of distance can be expressed: Given three locations l_1, l_2, l_3 we have that $distance(l_1, l_2) < distance(l_1, l_3)$ if $sup(\{l_1, l_2\}) < sup(\{l_2, l_3\})$ in the lattice. This may be a poor metric, but hierarchical models do not readily represent interconnections between locations.

Graph-based models

In the graph-based approach symbolic coordinates define the vertices V of graph $G = (V, E)$. An edge $e \in E$ is added between to vertices if a direct connection between those two vertices exists in the physical world. An edge could be a door between two rooms (vertices). Edges can be weighted (and oriented) to model distances. It is clear that this setup supports the connected-to and distance relationships explicitly. It is therefore well-suited for navigation and distance queries. The containment relationship is not supported, but can be simulated by linking from a reference vertex to all other vertices which are considered to be within a particular range. This is not a general mechanism though. Furthermore, locations can not consist of other locations. For examples, see "smart environments" [RLU94, OJDA01].

Combined graph- and set-based symbolic models

As seen, the set-based models support range queries well whereas the graph-based models support distance and connected-to. We wish to combine the two model types to obtain all the benefits. The set-based part of this hybrid model is a set of symbolic coordinates. Locations are sets of coordinates. Locations are connected by edges if a connection between these locations exists in the physical world. For instance, two rooms can be connected by a door, and two floors by a stairway. Edges can be weighted to model distances. We can introduce more than one graph to represent *views*, i.e., different aspects of the world such as physical and organisational. An example is the Active Map system [Sch95].

Geographical, semantic, and topological models

Geographical models typically organise the location space *hierarchically* via identifiers such as "City of Copenhagen", "The IT University of Copenha-

gen", and "The C wing".We consider these models to be a special case of symbolic models where the symbols happen to carry geographical meaning, and the location space is, e.g., organised as a tree.

We believe the term *semantic* location model was coined in [Pra00], where places (semantic locations) are represented by URIs and can have attributes indicating the nature or purpose, or even physical or geographical information. In [Pra00] there are three types of location; physical (grid based), geographical (hierarchical), and semantic (web like). Like in [Rot03], we do not distinguish semantic and geographical location. We consider semantic locations to be a special case of symbolic locations because web-like structures can well be described using graphs to model location space in a symbolic model. *Topological* [BS01] models are related to semantic models. Organising semantic locations in a hierarchy supports the containment relationship. Still, the combined model remains a special case of symbolic models.

2.3.3 Hybrid location models

Hybrid location models are so called because they are combinations of the two model types we have outlined above, namely symbolic and geometric. Symbolic and geometric models are orthogonal and can therefore be combined in numerous ways. Hybrid models aim to possess the advantages of both types of models, i.e., basically to provide applications with a high-level structured symbolic representation of locations while preserving the accuracy of location information inherent in geometric coordinates. This combination supports the queries we considered in Section 2.2. The trade-off is a higher modelling effort.

In [BD05] it is suggested to add geometric information to a symbolic model. This can be done either for each symbolic location, or for only some of them. It is also possible to deduce symbolic locations from geometric locations, and relative from absolute by the containment relation [HHS+02]. It is a matter of abstracting certain geometric data into meaningful symbolic notions. Typically, a non-hybrid model will be geometric and absolute, or symbolic and relative.

One example of a hybrid model is found in [JS02], where in a symbolic tree model each node (location) has geometric information as an attribute, and queries such as distance and containment are supported. Another example is found in [Rot03], where a domain model is presented. This hybrid model has mappings between local and global coordinates, and also between geometric and symbolic locations. Local coordinates can be

translated into global coordinates. Global geometric coordinates can then be translated into global symbolic coordinates, which in turn can be translated into global geometric *areas*. Such mappings are widely adopted in practise according to [BD05].

2.3.4 Views

An idea found in several papers [BD05, BBR02, BS01, Rot03] is that of having *views*, i.e., having multiple hierarchies, e.g., representing different views of an organisation; the building, employee relationships, access control *et cetera*. This thought also appears in theoretical work, see Chapter 7.

2.3.5 Location-aware systems

Several *location-aware systems* (or *location systems*) have been implemented demonstrating the feasibility of using location information in practise while challenging existing and developing new technology. By 'location system' we mean a computer system that via hardware sensors can track the physical location of objects (to be explained shortly), and is able to output this information in some suitable format. See [HB01] for a survey of location systems where they have been categorised according to their properties geometric/symbolic and absolute/relative. Location systems as such are not within the focus of this dissertation so we refrain from further discussion.

2.4 A model of a reflective building

We describe a reflective building as "one equipped with sensors, which continually transmit data of the building's occupancy to a monitor that maintains a data structure which faithfully records the occupancy."[1]. This is a very loose description. We come a little closer to more tangible properties in [Hop00], where a sentient building is said to support containment, proximity (near), and coordinate systems. Except for the coordinate systems it does seem that we can do with a very simple location model, e.g., a symbolic tree model. This is the starting point for this dissertation. We do, however, wish to model more realistic (complex) examples in our work so coordinates should be considered. We will come back to this in Chapter 5.

[1] http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/fp-movement.html

2.5 Concluding remarks

The reader should at this point have sufficient background knowledge to be able to understand the next chapters. The most important points are:

- Location models facilitate efficient development of location-aware applications.
- To support a broad range of applications a location model should support four query types (position, near, navigation, range), and thus be a hybrid model (geometric and symbolic). Queries rely on certain spatial relationships.
- A symbolic tree model is a good starting point for a reflective building.

We have studied and synthesised the literature on location models and digested it for the reader. Much can and has been written about different implementations of location-aware applications, but the details of those works are outside the scope of this dissertation. Likewise for positioning systems and sensor technologies underlying a location model. We focus on mathematical modelling.

The subject of the next chapter is modelling of context-aware systems in Bigraphs.

3

Bigraphical Models of Context-aware Systems

This chapter consists of a paper [BDE+06] published at the 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06). It was a collaborative effort between Lars Birkedal, Søren Debois, Thomas Hildebrandt, Henning Niss, and I. I made a proportional contribution both in the research and writing phase, which is evidenced by a co-author statement accompanying this dissertation. The paper has been insignificantly altered to match the layout of this dissertation, and a few additional lines of explanations have been admitted. Moreover, the appendices from the accompanying technical report [BDE+05] have been included.

Abstract

As part of ongoing work on evaluating Milner's bigraphical reactive systems, we investigate bigraphical models of *context-aware systems*, a facet of ubiquitous computing. We find that naively encoding such systems in Bigraphs is somewhat awkward; and we propose a more sophisticated modelling technique, introducing *Plato-graphical models*, alleviating this awkwardness. We argue that such models are useful for simulation and point out that for reasoning about such bigraphical models, the bisimilarity inherent in bigraphical reactive systems is not enough in itself; an equivalence between the bigraphical reactive systems themselves is also needed.

3.1 Introduction

The theory of *bigraphical reactive systems*, due to Milner and co-workers, is based on a graphical model of mobile computation that emphasises both locality and connectivity [JM04, Mil04c, Mil06a]. A bigraph comprises a place graph, representing locations of computational nodes, and a link graph, representing interconnection of these nodes. We give dynamics to Bigraphs by defining reaction rules that rewrite bigraphs to bigraphs; roughly, a bigraphical reactive system (BRS) is a set of such rules. Based on methods of the seminal [LM00a], any BRS has a labelled transition system, the behavioural equivalence (bisimilarity) of which is a congruence.

There are two principal aims for the theory of bigraphical reactive systems: (1) to model ubiquitous systems [Wei93], capturing mobile locality in the place graph and mobile connectivity in the link graph; and (2) to be a meta-theory encompassing existing calculi for concurrency and mobility. To date, the theory has been evaluated only with respect to the second aim: We have bigraphical understanding of Petri nets [Mil04b], π-calculus [Jen07, JM04, JM03], CCS [Mil06a], mobile ambients [Jen07], HOMER [BH06], and λ-calculus [Mil04c, Mil05b].

The present paper initiates the evaluation of the first aim. We investigate modelling of *context-aware systems*, a vital aspect of ubiquitous systems. A context-aware application is an application that adapts its behaviour depending on the context at hand [SAW94], interpreting "context" to mean the situation in which the computation takes place [DA00]. The canonical example of such a situation is the location of the device performing the computation; systems sensitive to location are called *location-aware*. As an

example, a location-aware printing system could send a user's print job to a printer close by. (For notions of context different from location, refer to [SBG99]; for large-scale practical examples, see [ACH+01].)

To observe changes in the context, context-aware systems typically include a separate context sensing component that maintains a model of the current context. Such models are known as context models [HIR02] or, more specifically, location models [BD05]. The above-mentioned location-aware printing system would need to maintain a model of the context that supports finding the printer closest to a given device. Such models are informal. There are only very few formal models of context-aware computing (refer to [Hen04] for an overview). We point out Context Unity [RJP04]; in spirit, our proposal is somewhat closer to process calculi than Context Unity is. However, Bigraphs differ from traditional process calculi in that we get to write our own reaction rules. In overall terms, our contribution is two-fold.

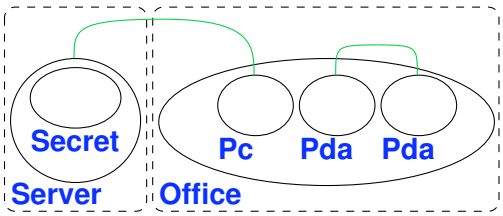
- We find, perhaps surprisingly, that naively modelling context-aware systems as BRSs is somewhat awkward; and
- we propose a more sophisticated modelling technique, in which the perceived and actual context are both explicitly represented as distinct but overlapping BRSs. We call such models *Plato-graphical*.

The remainder of this paper is organised as follows. In Section 3.2, we introduce Bigraphs and bigraphical reactive systems. In Section 3.3, we discuss naive bigraphical models of location-aware systems. In Section 3.4, we introduce our Plato-graphical models of context-aware systems. In Section 3.5, we present two example models. In Section 3.6, we discuss. Finally, in Section 3.7, we conclude and note future work.

3.2 Bigraphs and bigraphical reactive systems

We introduce Bigraphs by example (the reader can find the relevant formal definitions of [JM04, Mil06a] in Appendix A.1. Readers acquainted with Bigraphs may skip this section.

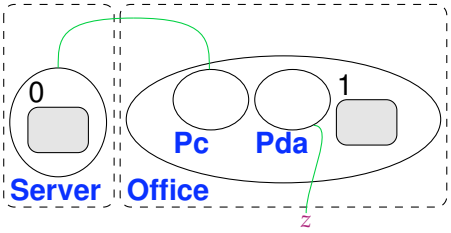
Here is a bigraph, A:



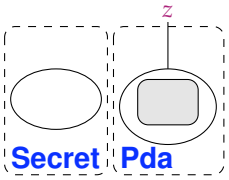
It has *nodes* (vertices), indicated by solid boxes. Each node has a *control*, written in sans serif. Each control has a number of *ports*; ports can be linked by *edges*, indicated by lines. Here, the controls secret and office have no ports, all other controls have one port. Nodes can be nested, indicated by containment. The two outermost dashed boxes indicate *roots*. Roots have no controls; they serve solely to separate different nesting hierarchies.

The bigraph A ostensibly models two physically separate locations (because of the two roots). The first contains a server, which in turn contains secret data; the second contains an office, which in turn contains a PC and two PDAs. The server and the PC are connected, as are the PDAs.

Here is another bigraph, B:



B resembles A, except that the content of server has been replaced with a *site* $-_0$, one of the pdas has been replaced by a site $-_1$, and there is an *inner name* z connected to the remaining pda. Using sites and names, we can define composition of bigraphs. For that, here is yet another bigraph C:



C has an *outer name* z. The bigraphs B and C compose to form A, i.e., $A = B \circ C$. Composition proceeds by plugging the roots of C into the sites of B (in order), and fusing together the connections pda $\to z$ (in C) and $z \to$ pda (in B) removing the name z in the process.

One cannot compose arbitrary bigraphs. For $U \circ V$ to be defined, U must have exactly as many sites as V has roots, and the inner names of U must be precisely the outer names of V. The sites and inner names are collectively called the *inner face*; similarly, the roots and outer names are called the *outer face*. A has inner face $\langle 0, \emptyset \rangle$ (no holes, no inner names) and outer face $\langle 2, \emptyset \rangle$ (two roots, no outer names). We write $A : \langle 0, \emptyset \rangle \to \langle 2, \emptyset \rangle$. Similarly, $B : \langle 2, \{z\} \rangle \to \langle 2, \emptyset \rangle$ and $C : \langle 0, \emptyset \rangle \to \langle 2, \{z\} \rangle$.

The graphical representation used above is handy for modelling, but unwieldy for reasoning. Fortunately, Bigraphs have an associated term language [DB05, Mil04a], which we use (albeit in a sugared form) in what follows. The language is summarised in Table 3.1. Here are, in order of

Term	Meaning
$U \parallel V$	Concatenation (juxtaposition) of roots.
$U \mid V$	Concatenation (juxtaposition) of children. (Collect the children of U and V under one root.)
$U \circ V$	Composition.
$U(V)$	Nesting: U contains V.
$\mathsf{K}_{\vec{x}}(U)$	Ion: Node with control K of arity $\lvert\vec{x}\rvert$, ports connected to the outer names of vector $\vec{x}$. The node contains U.
1	The *barren* (empty) root.
$-_i$	Site numbered i.
$/x.U$	U with outer name x replaced by an edge.
x/y	Connection from inner name y to outer name x.

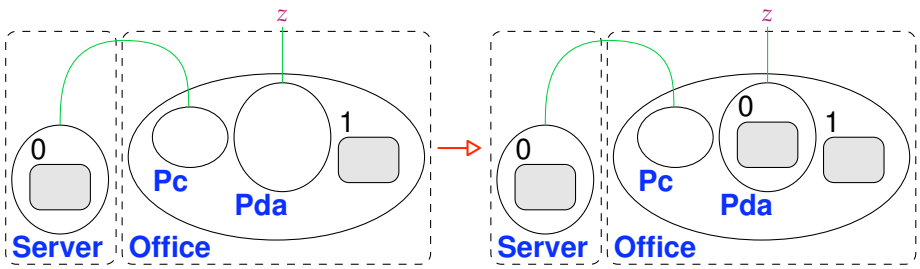
Table 3.1: Sugared term language for Bigraphs.

increasing complexity, term representations of the bigraphs A, B and C.

$$
\begin{aligned}
C &= \mathsf{secret} \parallel \mathsf{pda}_z \\
A &= /x./y.\mathsf{server}_x(\mathsf{secret}) \parallel \mathsf{office}(\mathsf{pc}_x \mid \mathsf{pda}_y \mid \mathsf{pda}_y) \\
B &= /x./y.\mathsf{server}_x(-_0) \parallel \mathsf{office}(\mathsf{pc}_x \mid \mathsf{pda}_y \mid -_1) \mid y/z
\end{aligned}
$$

Notice how, in B, edges are specified by first linking nodes to the same name, then converting that name to an edge using the closure '/'.

We give dynamics to Bigraphs by defining reaction rules. Example:



$$/x.\mathsf{server}_x(-_0) \parallel \mathsf{office}(\mathsf{pc}_x \mid \mathsf{pda}_z \mid -_1)$$
$$\rightarrowtriangle\ /x.\mathsf{server}_x(-_0) \parallel \mathsf{office}(\mathsf{pc}_x \mid \mathsf{pda}_z(-_0) \mid -_1)$$

This rule might model that if a PC in some office is linked to a server, a PDA in the same office may use the PC as a gateway to copy data from the server. The rule matches the bigraph A above, taking secret to the site $-_0$ and pda_y to the site $-_1$, rewriting A to

$$A' = /x./y.\mathsf{server}_x(\mathsf{secret}) \parallel \mathsf{office}(\mathsf{pc}_x \mid \mathsf{pda}_y(\mathsf{secret}) \mid \mathsf{pda}_y)$$

(We omit details on what it means to match connections; refer to one of [JM04, Mil06a].)

It is occasionally convenient to limit the contexts in which a reaction rule applies [BP04], i.e., we might want to limit the above example reaction rule to apply only in the left wing of the building. To this end, Bigraphs can be equipped with a *sorting* [Jen07, Mil06a, Mil04b]. A sorting consists of a set of *sorts* (or types); all inner and outer faces are then enriched with such a sort. Further, a sorting must stipulate some condition on bigraphs, we then restrict our attention to the bigraphs that satisfy that condition, thus outlawing some contexts. Obviously, removing contexts may ruin the congruence property of the induced bisimilarity; [Jen07] and [Mil06a] give different sufficient conditions for a sorting to preserve that congruence property.

This concludes our informal overview of Bigraphs. Now, on to the models.

3.3 Naive models of location-aware systems

In this section, we attempt to model location-aware systems naively in Bigraphs. We will find the naive approach to be somewhat awkward. Due to space constraints we do not discuss other forms of context.

We use the place and link graphs for describing locations and interconnections directly, and we use reaction rules to implement both *reconfiguration* of the context and *queries* on the context. The former is simply a non-deterministic change in the context; the latter is a computation on the context that does not change the context, except for producing an answer to some question. In a location-aware system, a device moving would be a reconfiguration, whereas computing the answer to the question "what devices are currently at the location l" is a query.

We discuss the implementation of this query. (An implementation of a "find all" query can be found in Section 3.A.) Incidentally, a query such

as "find nearest neighbour", which conceptually is only slightly harder, is significantly harder to implement. (Other examples plagued by essentially the same difficulties can be found in [DD05].)

Consider the following bigraph l representing devices (e.g. PDAs) located at locations (e.g. offices, meeting rooms) within a building.

$$/w./x./y./z.\mathsf{loc}\Big(\mathsf{loc}\Big(\mathsf{loc}\Big(\mathsf{loc}\,(\mathsf{devi}_w) \mid \mathsf{loc}\Big(\mathsf{devi}_x \mid \mathsf{devi}_y\Big)\Big)\Big) \mid \mathsf{loc}() \mid \mathsf{loc}\,(\mathsf{devi}_z)\Big)$$

Off-hand, finding all devices, say, beneath the root, looks straightforward: We should simply recursively traverse the nesting tree. Unfortunately, such traversal is quite complicated for the following reasons.

- The bigraphical reaction rules do not support recursion directly, so we must encode a runtime stack by means of additional controls.
- Bigraphical reaction rules can be applied in *any* context, but when implementing an operation such as the query we consider now, we need more refined control over when rules can be applied; one may achieve this more refined control by again using additional nodes and controls, essentially implementing what corresponds to a program counter. This still leaves great difficulty in handling concurrent operations, though.
- As a special case of the previous item, it is particularly difficult to express that a reaction rule is intended to apply only in case something is *not* present in the context.

Summing up, the bigraphical rules that model physical action do not in general provide the power to compute directly with a model of that action (because of a lack of control structures). The slogan is "reconfiguring is easy, querying is hard".

In earlier work on evaluating Bigraphs as a meta-theory (aim (2) mentioned in the Introduction), reaction rules were used to encode the operational semantics of a calculus or programming language. However, above we attempt to implement a query *directly* as reaction rules. This seemingly innocuous difference will turn out to have major implications for reasoning methods; more on this in Section 3.6.

We imagine that adding more flexibility to the reaction rules might make it easier to program directly with Bigraphs. One possible attempt is to use spatial logics for Bigraphs [CMS05] in combination with sorting, to get control of the contexts in which a particular reaction rule applies.

In the following sections, we propose another way to model context-aware systems in Bigraphs, where the reaction rules are not used to program directly with, but instead they are used (1) to represent transitions happening in the real world and (2) to encode operational semantics of programming languages, within which one can then implement queries on representations of the real world.

3.4 Plato-graphical models of context-aware systems

The naive model of the previous section shares an important characteristic with recent proposals of formal models for context-aware computation [BP04, NGP05, RJP04] that comprise agents and contexts only: These models take the agent's ability to determine *what is* the present context as given. We contend that for some systems, it is natural to model not only the actual context but also the agent's representation of the actual context. We shall see that pursuing this idea will partially alleviate the awkwardness seen in the previous section. We shall need some notation and definitions.

Notation 3.1. *We write* $\mathbf{B} = (\mathcal{K}, \mathcal{R})$ *to indicate that* $\mathbf{B}$ *is a bigraphical reactive system with controls* $\mathcal{K}$ *and rules* $\mathcal{R}$*, and write* $f \in \mathbf{B}$ *to mean that* f *is a bigraph of* $\mathbf{B}$*.*

Definition 3.2 (Independence)**.** *Let* $\mathbf{B} = (\mathcal{K}, \mathcal{R})$ *and* $\mathbf{B}' = (\mathcal{K}', \mathcal{R}')$ *be bigraphical reactive systems. Say that* $\mathbf{B}$ *and* $\mathbf{B}'$ *are* independent *and write* $\mathbf{B} \perp \mathbf{B}'$ *iff* $\mathcal{K}$ *and* $\mathcal{K}'$ *are disjoint.*

Definition 3.3 (Composite bigraphical reactive systems)**.** *Let* $\mathbf{B} = (\mathcal{K}, \mathcal{R})$ *and* $\mathbf{B}' = (\mathcal{K}', \mathcal{R}')$ *be bigraphical reactive systems. Define the union* $\mathbf{B} \cup \mathbf{B}'$ *point-wise, i.e.,* $\mathbf{B} \cup \mathbf{B}' = (\mathcal{K} \cup \mathcal{K}', \mathcal{R} \cup \mathcal{R}')$*, when* $\mathcal{K}$ *and* $\mathcal{K}'$ *agree on the arities of the controls in* $\mathcal{K} \cap \mathcal{K}'$*.*

Be aware that there are two ways of taking the union of two sets of *parametrised* reaction rules: (1) merge the rules and then ground them, or (2) first ground the rules and then merge them. In general, the resulting rule set of (1) is a superset of the rule set of (2), because parametric rules from one BRS may be instantiated with bigraphs from the other. We use approach (1).

We propose a model of context-aware computing that comprises three bigraphical reactive systems: the context $\mathbf{C}$; its representation or proxy $\mathbf{P}$; and the computational agents $\mathbf{A}$. Drawing on classical work [PlaBC], specifically The Allegory of the Cave, we call such a model *Plato-graphical*.

Definition 3.4 (Plato-graphical model)**.** *A* Plato-graphical model *is a triple* $(\mathbf{C}, \mathbf{P}, \mathbf{A})$ *of bigraphical reactive systems, such that* $\mathcal{M} = \mathbf{C} \cup \mathbf{P} \cup \mathbf{A}$ *is itself a bigraphical reactive system and* $\mathbf{C} \perp \mathbf{A}$*. A* state *of the model is a bigraph of* $\mathcal{M}$ *on the form* $/\vec{x}.(C \parallel P \parallel A)$*, where* $C \in \mathbf{C}$*,* $P \in \mathbf{P}$*,* $A \in \mathbf{A}$*, and* $\vec{x}$ *is some vector of names.*

We emphasise the intended difference between **C** and **P**: Whereas an element of **C** models a possible context, an element of **P** models *a model* of a possible context. The independence condition ensures that agents can only directly observe or manipulate the proxy; not the context itself. (In the parlance of [RJP04], the independence condition ensures separability.) To query or alter the context, agents must use the proxy as a sensor and actuator.

Using Bigraphs as our basic formalism gives us two things. First, we can write our own reaction rules. We claim that because of this ability, models become remarkably straightforward and intuitive; hopefully, the reader will agree after seeing our example models in the next section. Second, we automatically get a bisimilarity that is a congruence. Thus, bisimilarity of agents is a very fine equivalence: No state of the context and proxy can distinguish bisimilar agents.

The bisimilarity of the following proposition is the wide bisimilarity of Definition 5.3 in [JM04], which corresponds to classical strong bisimilarity of, e.g., π-calculus, i.e., the largest bisimulation (which is a congruential and symmetric simulation).

Proposition 3.5. *Let* $\sim$ *denote the bisimilarity in* $\mathcal{M}$*, and let* $A, A' \in \mathbf{A}$ *with* $A \sim A'$*. For any* $C \in \mathbf{C}$*,* $P \in \mathbf{P}$*, and* $\vec{x}$*, we have* $/\vec{x}.(C \parallel P \parallel A) \sim /\vec{x}.(C \parallel P \parallel A')$*.*

To get a less discriminating equivalence we can consider agents under a particular state of the context, or a particular state of the system.

Definition 3.6. *Let* $\sim$ *denote the bisimilarity in* $\mathcal{M}$*, and let* $A, A' \in \mathbf{A}$*,* $C \in \mathbf{C}$ *and* $P \in \mathbf{P}$*. We say* A *and* A' *are* equivalent wrt. P *iff* $P \parallel A \sim P \parallel A'$*, and we say* A *and* A' *are equivalent wrt.* C, P *iff* $C \parallel P \parallel A \sim C \parallel P \parallel A'$*.*

We conjecture that the above forms of derived equivalences will prove useful for reasoning about a given Plato-graphical system.

Working within the Plato-graphical model, we are free to emphasise any of its three components, perhaps modelling **P** in great detail, but keeping **C** and **A** abstract.

Definition 3.4 above does not impose any restriction on composition of states. For example, assume that we have a Plato-graphical model $\mathcal{M} =$

($\mathbf{C}, \mathbf{P}, \mathbf{A}$), that c, p and a are controls of $\mathbf{C}$, $\mathbf{P}$ and $\mathbf{A}$, respectively, and that p is *not* a control of $\mathbf{C}$. Then the bigraphs

$$F = \mathsf{c}(-0 \mid -1) \parallel \mathsf{p} \parallel \mathsf{a}(-2) \qquad \text{and} \qquad G = \mathsf{c} \parallel \mathsf{p} \parallel \mathsf{a}$$

are both states of $\mathcal{M}$, but their composite $F \circ G = \mathsf{c}(\mathsf{c} \mid \mathsf{p}) \parallel \mathsf{p} \parallel \mathsf{a}(\mathsf{a})$ is not a state of $\mathcal{M}$. This example implies that bisimilarity of states of a Plato-graphical system may be too fine a relation: Conceivably, when comparing two states s and s', we may wish to take into account only contexts C such that $C \circ s$ and $C \circ s'$ are themselves states, i.e., we might want to outlaw F as a possible context for G. We can achieve this finer control using place-sorting. So, we define a place-sorted Plato-graphical model. The intuition behind our sorting is that we want to keep controls of $\mathbf{C}$, $\mathbf{P}$ and $\mathbf{A}$ separate when composing contexts of form $\mathbf{C} \parallel \mathbf{P} \parallel \mathbf{A}$.

Notation 3.7. *Denote by $S_{i\leq m}$ a vector $m_0, \ldots, m_{n-1}$ of sorts. We will write $S_{i\leq m}$ for a sorted interface $\langle m, X, S_{i\leq m}\rangle$ when we do not care about names.*

Definition 3.8 (Sorted Plato-graphical model)**.** *Let $\mathcal{M} = \mathbf{C} \cup \mathbf{P} \cup \mathbf{A}$ be a Plato-graphical model with $\mathbf{C} = (\mathcal{K}_\mathbf{C}, \mathcal{R}_\mathbf{C})$, $\mathbf{P} = (\mathcal{K}_\mathbf{P}, \mathcal{R}_\mathbf{P})$ and $\mathbf{A} = (\mathcal{K}_\mathbf{A}, \mathcal{R}_\mathbf{A})$. Define a sorting discipline on $\mathcal{M}$ by taking sorts $\Theta = \{\mathcal{K}_\mathbf{C}, \mathcal{K}_\mathbf{P}, \mathcal{K}_\mathbf{A}\}$ and, for primes, sorting condition $\Phi(f : S_{i\leq n} \to S) = \mathit{ctrl}(f) \subseteq S \wedge \forall i \leq n.\ S_i = S$, lifting to an arbitrary bigraph f' by decomposing f into primes $f' = f_0 \ldots f_{n-1}$ and declaring f' well-sorted iff all the f_i are. Let ϕ be an assignment of Θ-sorts to the rules of $\mathcal{R}_\mathbf{C}$, $\mathcal{R}_\mathbf{P}$, and $\mathcal{R}_\mathbf{A}$, such that every rule is well-sorted under Φ. Define $\mathcal{M}'$ to be $\mathcal{M}$ sorted by (Θ, Φ) (using ϕ to lift the reaction rules). In this case, we call $\mathcal{M}'$ a* sorted Plato-graphical model*, and define the states of $\mathcal{M}'$ to be the well-sorted bigraphs with outer face $\mathcal{K}_\mathbf{C}, \mathcal{K}_\mathbf{P}, \mathcal{K}_\mathbf{A}$.*

The condition Φ essentially requires that (1) the controls of a prime (bigraph) are elements of the sort of its outer face, and (2) the sort of the outer face is exactly the sort of each of the sites. Under this sorting discipline and new definition of state, if G is assigned a sort such that it is a state, then F cannot be assigned a sort that makes it composable with G.

Is the bisimilarity in the sorted system $\mathcal{M}'$ a congruence? The sorting discipline of $\mathcal{M}'$ is in general not homomorphic in the sense of Milner [Mil06a, Definition 10.4]: we cannot give a sort to controls in $\mathcal{K}_\mathbf{C} \cap \mathcal{K}_\mathbf{P}$. (If $\mathbf{C}$, $\mathbf{P}$ and $\mathbf{A}$ are pairwise independent, the sorting is homomorphic; however, such a model is pathological.) Neither is the sorting safe in the sense of Jensen [Jen07, Definition 4.30]; condition (4) cannot be met. Counterexample: Suppose $f : \mathcal{K}_\mathbf{C} \to \mathcal{K}_\mathbf{C}$ is well-sorted; take $g = f \otimes 1 : \mathcal{K}_\mathbf{C} \to \mathcal{K}_\mathbf{C}, \mathcal{K}_\mathbf{A}$

(recall that $1 : \epsilon \to \langle 1, \emptyset \rangle$ denotes the barren root). Clearly, $\mathcal{U}(f) = (-_0 \mid -_1) \circ \mathcal{U}(f \otimes 1)$. However, if $\mathcal{K}_\mathbf{C} \neq \mathcal{K}_\mathbf{A}$ then $(-_0 \mid -_1) : \mathcal{K}_\mathbf{C}, \mathcal{K}_\mathbf{A} \to \mathcal{K}_\mathbf{C}$ is not well-sorted.

Nevertheless, the sorting of definition 3.8 does give rise to a bisimilarity that is a congruence; we prove so in Appendix C.

3.5 Examples

3.5.1 A simple context-aware printing system

We model the simple context-aware printing system of [BP04]. An office-building contains both modern PCL-5e compatible printers and old-fashioned raw-printers. Occasionally, the IT-staff at the building removes or replaces either type of printers. Each printer can process only one job; queueing is done by a central print server. The print server dispatches jobs to raw-printers only if it knows no PCL-printers; if there are PCL-printers, but they are all busy, the job will simply have to wait. This system is context-aware: The type and number of printers physically available determine the meaning of the action "to print". We give a model **B** of this system in Figure 3.1. Looking at the controls of **B**, it is straightforward to verify that **B** is Plato-graphical.

Proposition 3.9. *The model* **B** *of Figure 3.1 is Plato-graphical.*

We take a detailed look at the model. A state of the context **C** consists of nested physical locations loc, within which printers prt are placed. We distinguish between PCL- and raw-printers by putting a token pcl and raw within them, respectively. Each printer has a single port, intended to link the printer to the proxy. Here is a state of the context with a PCL-printer and a raw-printer at adjacent locations; the PCL-printer is idle whereas the raw-printer is busy.

$$C = \mathsf{loc}(\mathsf{loc}(\mathsf{prt}_x(\mathsf{raw} \mid \mathsf{dat}_z)) \mid \mathsf{loc}(/y.\mathsf{prt}_y(\mathsf{pcl}))) \ .$$

Setting C in parallel with some proxy P will allow P access to the raw printer through the shared link x, but not to the PCL-printer, because it is in a closed link. The dynamics of **C** allow printers to appear (3.1, 3.2), disappear (3.3), and finish printing (3.4).

A state of the proxy **P** consists of a pool of pending jobs jobs and two tables of printers prts; one contains a token raw, the other a token pcl, indicating what type of printer the table lists. The prts is a table in the sense that its only port is linked to all the printers in the context that the

Context **C**.

Control	Activity	Arity	Comment
loc	active	0	Nested location
prt	passive	1	Physical printer
pcl	atomic	0	Printer-type token
raw	atomic	0	Printer-type token
dat	atomic	1	Binary data for printer

$$\mathsf{loc}(-_0) \;\rightarrow\; \mathsf{loc}(-_0 \mid /x.\mathsf{prt}_x(\mathsf{raw})) \tag{3.1}$$

$$\mathsf{loc}(-_0) \;\rightarrow\; \mathsf{loc}(-_0 \mid /x.\mathsf{prt}_x(\mathsf{pcl})) \tag{3.2}$$

$$\mathsf{loc}(-_0 \mid \mathsf{prt}_x(-_1)) \;\rightarrow\; \mathsf{loc}(-_0) \mid x/ \tag{3.3}$$

$$\mathsf{prt}_x(\mathsf{dat}_z \mid -_0) \;\rightarrow\; \mathsf{prt}_x(-_0) \mid z/ \tag{3.4}$$

Proxy **P**.

Control	Activity	Arity	Comment
prt	passive	1	Physical printer
pcl	atomic	0	Printer-type token
raw	atomic	0	Printer-type token
dat	atomic	1	Binary data for printer
prts	passive	1	Known devices
jobs	passive	0	Pending documents
doc	atomic	1	Document

$$\begin{aligned}\mathsf{jobs}(\mathsf{doc}_z \mid -_0) \parallel \mathsf{prts}_y(\mathsf{pcl}) \parallel \mathsf{prt}_y(\mathsf{pcl}) \rightarrow \\ \mathsf{jobs}(-_0) \parallel \mathsf{prts}_y(\mathsf{pcl}) \parallel \mathsf{prt}_y(\mathsf{pcl} \mid \mathsf{dat}_z)\end{aligned} \tag{3.5}$$

$$\begin{aligned}\mathsf{jobs}(\mathsf{doc}_z \mid -_0) \parallel /x.\mathsf{prts}_x(\mathsf{pcl}) \mid \mathsf{prts}_y(\mathsf{raw}) \parallel \mathsf{prt}_y(\mathsf{raw}) \rightarrow \\ \mathsf{jobs}(-_0) \parallel /x.\mathsf{prts}_x(\mathsf{pcl}) \mid \mathsf{prts}_y(\mathsf{raw}) \parallel \mathsf{prt}_y(\mathsf{raw} \mid \mathsf{dat}_z)\end{aligned} \tag{3.6}$$

$$/x.\mathsf{prt}_x(\mathsf{pcl}) \parallel \mathsf{prts}_y(\mathsf{pcl}) \;\rightarrow\; \mathsf{prt}_y(\mathsf{pcl}) \parallel \mathsf{prts}_y(\mathsf{pcl}) \tag{3.7}$$

$$/x.\mathsf{prt}_x(\mathsf{raw}) \parallel \mathsf{prts}_y(\mathsf{raw}) \;\rightarrow\; \mathsf{prt}_y(\mathsf{raw}) \parallel \mathsf{prts}_y(\mathsf{raw}) \tag{3.8}$$

Agents **A**.

Control	Activity	Arity	Comment
jobs	passive	0	Pending documents
doc	atomic	1	Document

$$\mathsf{jobs}(-_0) \;\rightarrow\; \mathsf{jobs}(-_0 \mid /z.\mathsf{doc}_z) \tag{3.9}$$

Figure 3.1: Example Plato-graphical model **B**.

Context **C**	Proxy **P**	Agent **A**
(3.1) : $\mathcal{K}_{\mathbf{C}}$	(3.5) : $\mathcal{K}_{\mathbf{A}}, \mathcal{K}_{\mathbf{P}}, \mathcal{K}_{\mathbf{C}}$	(3.9) : $\mathcal{K}_{\mathbf{A}}$
(3.2) : $\mathcal{K}_{\mathbf{C}}$	(3.6) : $\mathcal{K}_{\mathbf{A}}, \mathcal{K}_{\mathbf{P}}, \mathcal{K}_{\mathbf{C}}$	
(3.3) : $\mathcal{K}_{\mathbf{C}}$	(3.7) : $\mathcal{K}_{\mathbf{C}}, \mathcal{K}_{\mathbf{P}}$	
(3.4) : $\mathcal{K}_{\mathbf{C}}$	(3.8) : $\mathcal{K}_{\mathbf{C}}, \mathcal{K}_{\mathbf{P}}$	

Figure 3.2: Sorts for the rules of **C**, **P**, and **A**.

table knows about. Here is an example state of the proxy which knows one raw-printer, knows no PCL-printers and has two pending jobs.

$$P = \mathsf{prts}_x(\mathsf{raw}) \mid /y.\mathsf{prts}_y(\mathsf{pcl}) \mid \mathsf{jobs}(/z.\mathsf{doc}_z \mid /z'.\mathsf{doc}_{z'})$$

Setting C and P above in parallel by $\|$, and closing the link x, we get a system $/x.C \parallel P$, where the table $\mathsf{prts}_x(\mathsf{raw})$ and the physical printer $\mathsf{prt}_x(\mathsf{raw} \mid \mathsf{dat})$ are linked. The dynamics of **P** state that if there is a job and a known, idle PCL-printer, the proxy may activate this printer (3.5); that if there is a job, no known PCL-printer, and an idle raw-printer, the context may activate that printer (3.6); and finally, that the proxy may discover a previously unknown printer (3.7, 3.8).

The dynamics of **A** allow the agents to spontaneously spool documents (3.9).

Notice how the two printing rules (3.5) and (3.6) do not observe the context directly. Instead, the proxy observes the context (rules (3.7) and (3.8)) and records its observations in the tables $\mathsf{prts}_x(\mathsf{raw})$ and $\mathsf{prts}_y(\mathsf{pcl})$; the printing rules (3.5) and (3.6) then consult the tables. It is straightforward to determine whether there are no known PCL-printers: simply check if the table of PCL-printers has the form $/y.\mathsf{prts}_y(\mathsf{pcl})$.

As observed in section 3.3 and [BP04], it is generally very difficult, if not impossible, to observe the *absence* of something in the context directly. An interesting but rather natural consequence of the indirect observation is that it becomes asynchronous, i.e., it is possible that a PCL-printer exists but has not yet been observed.

This model **B** can be lifted to a sorted one by adding the sorts given in Figure 3.2; the figure assigns sorts to the outer face of both the redeces and reactums of the indicated rules. It is straightforward to verify that all of the rules are well-sorted.

Proposition 3.10. *The model* **B** *with the sorting assignment of Figure 3.2 is a sorted Plato-graphical model.*

3.5.2 *A location-aware printing system*

Suppose we extend the printing system with location-awareness, by stipulating that a print job is not printed until the printer and the device submitting the job are co-located. To model this extended system, we introduce a new control devi for devices (PCs or PDAs) with one port and change doc to include an extra port so we can link submitted jobs to the devices submitting them. The linking is reflected in the following modified rule (3.9) for spooling print jobs:

$$\mathsf{loc}(\mathsf{devi}_x \mid -_0) \parallel \mathsf{jobs}(-_1) \rightarrowtriangle \mathsf{loc}(\mathsf{devi}_x \mid -_0) \parallel \mathsf{jobs}(-_1 \mid /z.\mathsf{doc}_{z,x}) \qquad (3.9')$$

We must also modify rules (3.5) and (3.6) to insist that the device and printer are co-located. Rule (3.5) becomes

$$\begin{aligned} \mathsf{jobs}(\mathsf{doc}_{z,x} \mid -_0) \parallel \mathsf{prts}_y(\mathsf{pcl}) \parallel \mathsf{loc}(\mathsf{devi}_x \mid \mathsf{prt}_y(\mathsf{pcl})) \rightarrowtriangle \\ \mathsf{jobs}(-_0) \parallel \mathsf{prts}_y(\mathsf{pcl}) \parallel \mathsf{loc}(\mathsf{devi}_x \mid \mathsf{prt}_y(\mathsf{pcl} \mid \mathsf{dat}_z)). \end{aligned} \qquad (3.5')$$

(We suppress the new rule (3.6').)

Modifying the system once again, instead of insisting that device and printer have to be actually co-located, we just require the print job to end at a printer close to the device. The print server will need to query the proxy for the printer nearest a given device. We saw in subsection 3.3 that implementing such queries is awkward, so we will need to use the proxy. In the preceding section, we did so directly in Bigraphs; this time around, we transfer the expressive convenience of a general-purpose programming language to Bigraphs for ease of implementation. We use Bigraphs directly for modelling the actual context $\mathbf{C}$, whereas we will exploit Bigraphs as a meta-calculus for modelling the proxy $\mathbf{P}$.

In detail, the whole model is $\mathbf{B} = \mathbf{C} \cup \mathbf{P} \cup \mathbf{A}$, with $\mathbf{P} = \mathbf{S} \cup \mathbf{L}$. Here $\mathbf{C}$ is intended to be a bigraphical model of the "real world", the proxy $\mathbf{P}$ is comprised of a location sensor $\mathbf{S}$ and a location model $\mathbf{L}$, and $\mathbf{A}$ is the location-based application (the "computational agent").

A state C of $\mathbf{C}$ could look like this:

$$C = \mathsf{loc}(\mathsf{loc}(\mathsf{loc}(\mathsf{loc}(\mathsf{devi}_w) \mid \mathsf{loc}(\mathsf{devi}_x \mid \mathsf{devi}_y))) \mid \mathsf{loc} \mid \mathsf{loc}(\mathsf{devi}_z))$$

Changes in the real world are modeled by reaction rules that reconfigure such states. If we want to model, say, that a device may move from one location to another, we include the reaction rule

$$\mathsf{loc}(\mathsf{devi}_x \mid -_0) \parallel \mathsf{loc}(-_1) \rightarrowtriangle \mathsf{loc}(-_0) \parallel \mathsf{loc}(\mathsf{devi}_x \mid -_1). \qquad (3.10)$$

To implement the proxy, encode as a BRS a programming language $\mathcal{L}$ with data structures, communication primitives, and concurrency, e.g., Pict [PT00] or CML [Rep99]. (We return to this assumption below.) That is, define a translation from terms of $\mathcal{L}$ to Bigraphs, and add reaction rules encoding the operational semantics of $\mathcal{L}$. *Then implement the location model, the sensor, and the agents in $\mathcal{L}$ and use the encoding to transfer that model to Bigraphs*. In particular, a state of the location model **L** will have a data structure *representing* the current state of **C**. If $\mathcal{L}$ is an even half-way decent programming language, it should be straightforward to implement queries such as one of section 3.3 or the "find closest printer" we need above.

The sensor informs the location model about changes in **C**. We extend the above rule (3.10) moving a device to

$$(\mathsf{loc}(\mathsf{devi}_x \mid -_0) \parallel \mathsf{loc}(-_1)) \mid S \mid L \twoheadrightarrow (\mathsf{loc}(-_0) \parallel \mathsf{loc}(\mathsf{devi}_x \mid -_1)) \mid S' \mid L, \quad (3.10')$$

where S' is an $\mathcal{L}$-encoding of "send a notification to **L** that device x has moved". Upon receiving the notification, **L** updates its representation of the world. Agents of **A** can in turn query **L** when they need location information.

3.6 Discussion

We consider the following questions.

1. What languages $\mathcal{L}$ can we encode?
2. How close are Plato-graphical models to real systems?
3. What challenges have we found for bigraphical models?
4. What uses do we envision for Plato-graphical models?
5. How do we reason about Plato-graphical models?

Ad. 1. As mentioned, there exist bigraphical encodings of various π-calculi [Jen07, JM04, JM03] and of the λ-calculus [Mil04c, Mil05b]. Using ideas of the latter encodings, we have encoded MiniML (call-by-value λ-calculus with pairs and lists) in Local Bigraphs [Mil04c]. Based on our experiences with this encoding, we find it palatable to encode CML or Pict[1].

[1] We are presently working on implementing an interpreter for bigraphical reactive systems; such an interpreter will make it easier to experiment with these and other encodings.

Ad. 2. The model closely reflects how some actual location-aware systems work, for instance the one running at the ITU. Here, a sensor system (made by Ekahau) computes every two seconds the physical location of every device on the WLAN. The sensor system informs a location model about updates to locations; location-aware services then interact with the location model. In our sketched Plato-graphical model, the location model **L** may lag behind the actual **C**, if **L**'s representation of **C** does not reflect some recent reconfiguration of **C**. But that also happens in the real system at the ITU – when a location-aware service asks the location model for the whereabouts of a device, it obtains not the position of the device, but the position of the device the last time the sensor checked. In the mean time, the device may have moved.

Ad. 3. When modelling the physical world, we have made use of both the place and link graphs, the place graph modelling the location hierarchy of a building. As argued in [BD05], DAGs or graphs are more natural models of location. Thus, systems such as the ones we have considered here suggest generalising the place graph part of Bigraphs, or consider ways to encode DAGs or general graphs naturally as place graphs.

Ad. 4. Given an implementation of bigraphical reactive systems, one could *simulate* the behaviour of a location-aware system, and thus allow for experimentation with different designs of location-aware and context-aware systems. Likewise, one could experiment with different choices for the $\mathcal{L}$ language of Section 3.5.2. Such simulation suggests further extensions of the bigraphical model: In actual context-aware systems, one is generally interested in timing aspects (e.g. the sensor samples only every two seconds), continuous space (e.g. the sensor really produces continuous data), and probabilistic models (e.g. to accurately simulate sensors and sensor failure).

Ad. 5. What about using Plato-graphical models for *formal reasoning* about context-aware systems? One use of formal models is to prove an abstract specification model equivalent to a concrete implementation model. In π-calculus, we come with π-terms i, s, one for the implementation and one for the specification. The terms i and s are themselves the models; we take (π-)bisimilarity as equivalence, so to prove i and s equivalent, we merely prove them bisimilar. We can play the same game within any BRS: Simply come up with a bigraph I (the implementation model) and a bigraph S (the specification model), and prove them bisimilar within the labelled transition system of the BRS. Because that bisimulation is congruential, such reasoning should be tractable, e.g. with the bisimulation in Definition 3.6.

Unfortunately, bisimulation within a single BRS is not always enough with respect to Plato-graphical models. Suppose we want a specification model $\mathcal{M}$ with an abstract view of the context, and an implementation model $\mathcal{M}'$ with a detailed view of the context. We express this by having $\mathcal{M}$ and $\mathcal{M}'$ differ only in their context sub-BRSs, i.e.,

$$\mathcal{M} = \mathbf{C} \cup \mathbf{P} \cup \mathbf{A} \qquad \mathcal{M}' = \mathbf{C}' \cup \mathbf{P} \cup \mathbf{A}.$$

The trouble is that because $\mathbf{C}$ and $\mathbf{C}'$ may have different controls and reaction rules, bisimulation between their respective labelled transition systems is meaningless! What we need is a notion of equivalence of BRSs, not just equivalence of bigraphs of a single BRS. At the time of writing, we know of no such equivalence[2]. Thus, our investigation of bigraphical models for context-aware systems suggests that equivalence of BRSs is a key notion currently missing. One possible direction would be to try to recover from the notion of WRS-functor [LM00a] – functors that preserve reaction rules – a notion of a BRS implementing another BRS.

3.7 Conclusion and future work

We have initiated an evaluation of the use of bigraphical reactive systems for models of context-aware computing in ubiquitous systems. We found that BRSs, in their current form, are not suitable for directly modelling context queries, but on the other hand suitable for modelling reconfigurations of the actual context.

In response, we proposed Plato-graphical models, where both state and dynamics are logically divided in three parts: the actual context, the observed context (or proxy), and the computational agents, respectively. The computational agents and the actual context are separated, and interact only through the proxy. This separation into different BRSs makes it possible to encode different parts of the system using domain-specific languages. Moreover, we have shown how the context-aware printing system of [BP04] can be modelled straightforwardly in the Plato-graphical model.

Further, we have argued that Plato-graphical models are useful for simulating context-aware systems, and we are currently working on an implementation of BRSs at ITU to allow such experimentation. Only through

[2]The reader may suggest that we just define a common language for modelling both the abstract and detailed view, and define a translation from this language into a single BRS. However, in this case we are no longer modelling a ubiquitous system directly in Bigraphs (aim 1 of the Introduction), but using Bigraphs as a meta-calculus (aim 2 of the Introduction).

such experimentation will it be clear how useful Plato-graphical models really are. For simulation purposes it will be important to extend Bigraphs with timing aspects, continuous space, and probabilities.

Finally, we have pointed out that establishing a notion of equivalence between BRSs, as opposed to bisimilarity within a BRS, is important future work.

3.8 Acknowledgments

We gratefully acknowledge discussions with the other members of the BPL group at ITU, in particular Arne Glenstrup, Troels Damgaard and Mikkel Bundgaard; and with Robin Milner. This work was funded in part by the Danish Research Agency (grant no.: 2059-03-0031) and the IT University of Copenhagen (the LaCoMoCo project).

What follows is two sections that were appendices to the paper, and which feature in the corresponding technical report [BDE+05]. The first of these sections illustrates how one may program a simple query on a location model. The second section formally defines rigid control-sortings and prove that they are "safe", i.e., respect RPOs (relative pushouts).

3.A Encoding of "find all devices"

This section is due to Ebbe Elsborg.

Consider the following simple bigraph representing a building consisting of locations (e.g. rooms) and devices (e.g. PDAs) in these locations. (We have omitted the outer names on the locations, and also sites.)

$$l = \mathsf{loca}(\mathsf{loca}(\mathsf{loca}(\mathsf{loca}(\mathsf{devi}_1) \mid \mathsf{loca}(\mathsf{devi}_2 \mid \mathsf{devi}_3))) \mid \mathsf{loca}() \mid \mathsf{loca}(\mathsf{devi}_4))$$

Consider how to implement a query to return all the devices in the building by means of bigraphical reaction rules. Observe that we have chosen to represent all locations via the same control loca, rather than using different controls office, building, etc., for different locations – this is to avoid having to write reaction rules for every combination of location controls. (loca differs from loc in that it has a port.)

Now, assume that a query occurs by some process introducing a node with control f into the system (in a unique[3] in node), and that no other queries impose themselves while we calculate the answer to this one. The

[3]Certain properties can only be ensured by invariants of the reactive system, e.g. uniqueness of controls.

termination condition (observable by the "input/output process") is when in is empty (and nodes with control f′ appear in the node with unique control out). We can not handle concurrent queries so that is why we wish to detect termination (so that we can begin the next query).

The idea is to do a depth-first search/collection while keeping track of where we have already looked by placing these subtrees into "searched-nodes" (s). Controls:

Control	Activity	Arity	Comment
in	passive	0	Input node
f	atomic	0	Controls find-all query
f′	atomic	1	Answer node
loca	active	1	Nested location
out	passive	0	Output node
g	atomic	0	Dummy, just to keep in non-empty
devi	atomic	1	Device, has link to id
s	passive	0	Collects searched nodes

And now, for the rules. Initialisation; move f into the *top* location (enclosing all the others) to indicate "the point of control" in the structure, and add g to indicate that we are not done with the query:

$$\mathsf{in(f)} \mid \mathsf{loca}_{top}(\mathsf{loca}_x(-_0)) \mid \mathsf{out}() \rightarrow \mathsf{in(g)} \mid \mathsf{loca}_{top}(\mathsf{loca}_x(\mathsf{f} \mid \mathsf{s}() \mid -_0)) \mid \mathsf{out}()$$

If a device is found, add it to s, and add a representative for it to out:

$$\begin{aligned} &\mathsf{loca}_x(\mathsf{f} \mid -_0 \mid \mathsf{devi}_y \mid \mathsf{s}(-_1)) \mid \mathsf{out}(-_2) \\ \rightarrow\ &/y.\mathsf{loca}_x(\mathsf{f} \mid -_0 \mid \mathsf{s}(-_1 \mid \mathsf{devi}_y)) \mid \mathsf{out}(-_2 \mid \mathsf{f}'_y) \end{aligned}$$

Notice that the label (context) of this transition will include the bigraph top/x. This rule can be used as long as there are devices in the current location being searched. When done with this location f is moved up, since we assume that a location can only contain either devices or other locations. (The query would have been significantly harder without this assumption.) So, if a location containing location(s) is being searched:

$$\mathsf{loca}_x(\mathsf{f} \mid \mathsf{loca}_y(-_1) \mid \mathsf{s}(-_2) \mid -_0) \rightarrow \mathsf{loca}_x(\mathsf{loca}_y(\mathsf{f} \mid \mathsf{s}() \mid -_1) \mid \mathsf{s}(-_2) \mid -_0)$$

Then, search deeper. A new s-node is created when going down, this is a trick to do "on-line garbage collection" when climbing back up the tree. (The query has to leave the bigraph as it was initially.) When a leaf is reached (an empty location) move f up, merging s-nodes. This is more

clever than doing a clean-up traversal because there is no construction in reaction rules that can express "not", i.e., one can not write a rule saying "clean up until there are no more s-nodes in the tree". The "climbing" rule:

$$\mathsf{loca}_x(\mathsf{loca}_y(\mathsf{f} \mid \mathsf{s}(-_2)) \mid \mathsf{s}(-_1) \mid -_0) \rightarrow \mathsf{loca}_x(\mathsf{f} \mid \mathsf{s}(\mathsf{loca}_y(-_2) \mid -_1) \mid -_0)$$

Notice how the presented rules use sites to make themselves general. To clean up when we have traversed the whole tree (and there is exactly one s):

$$\mathsf{in}(g) \mid \mathsf{loca}_{top}(\mathsf{loca}_x(\mathsf{f} \mid \mathsf{s}(-_0))) \rightarrow \mathsf{in}() \mid \mathsf{loca}_{top}(\mathsf{loca}_x(-_0))$$

At this point out will have all representatives, in is emptied to indicate termination. (The reader is encouraged to try out the rules on the example location model. It is easiest doing it using the graphical bigraph notation.)

3.B Native queries

This section demonstrates, by an example, the difficulty of programming directly with bigraphical reaction rules. We represent a location model and the queries directly in pure Bigraphs. Before developing the model we consider some issues of representation.

We model buildings, floors, rooms and so forth by introducing 'containers'. This generality makes the representation more elegant, and more information about a container can be given by simply adding a link, if desired.

Containers can be empty, contain other container(s), or contain device(s) at any given point in time. Specifically, a container cannot contain both a container and a device at the same time. By 'leaves' we denote containers that do not contain other containers.

Regarding topology (place graph): The one tree representing the location model is placed as only child of a 'root' node. This is done to be able to recognise that a search is completed. The whole system is a forest.

Assume that there is some process that creates queries by introducing nodes with certain controls into the existing model of the system, that is, an interface between the user and the model. The algorithm is as follows:

1. Exactly one query node is inserted into the "global" (global means sibling of the unique root container) in control to avoid interference with the queries.

2. Calculate.

3. Output is likewise put in the "global" out control for immediate reading by "the IO process" – and in cleared. We use online garbage collection.

Queries are of two kinds: "Reconfiguring" and "asking". Reconfiguring queries alter the place graph, asking queries do not. The rewrite rules are constructed such that they get stuck when the termination condition is met (see below). Also, we apply dichotomy so that all cases are covered.

Regarding the "nearest neighbour" query, we look in same container first, if unsuccessful we do a depth-first search. We do not use the distance links of the range queries, since we are not yet sure whether it is the right way to do range queries. (Nearest neighbour was encoded before the range query.) The controls a and b are used to make sure that one set of rewrite rules is executed before another – this is needed to separate search from clean-up.

A tentative way of incorporating distance is to assume 'distance links' between all containers that do *not* contain other containers (these containers also have 0-links to themselves). These links represent the distance between the centres of two containers – this is realistic with respect to Bluetooth technology. Each distance link is connected, through the outer face, to an "integer name", for instance, '7'.

We want to be able to enquire about devices of certain types (without having to write rules for each pair). Thus, we need subtyping. We do this by assuming a global control types (sibling to root) which, for all pairs (t, t') of device types, holds either a control leq(left, right) where t is linked to left and t' to right, or gt(left, right) where t is linked to left and t' to right. Intuition: The larger the type, the more general. If we want to find all devices we just ask for devices with type 'top', which is greater than all other types in control types. Subtyping could also have been done by having a "types tree" to traverse (using a similar technique as in the nearest neighbour query).

The termination condition (when the IO process can report back to the user) for the queries is that 'in' is empty. Also, we assume some meta-conditions/invariants (see encoding).

```
signature Locmod =
  sig  % Arity: Number of ports. Take l.u.b. of req. ports in
       % queries to use fewest different controls.
    a : atomic (0) % exists while still searching
    b : atomic (0) % exists when found a dev, enable clean-up
    c : active (2) % container. ports: id,distance
```

```
    d : atomic (2) % device. ports: id,type
    d' : atomic (2) % device, answer. ports: id,id
    e : atomic (0) % exists when 'near' query may terminate
    f : atomic (1) % find all. ports: type
    f' : atomic (2) % find all, one answer. ports: id,type
    g : atomic (0) % dummy control for termination
    in : passive (0) % input, one query at a time
    left : atomic (1) % left part of binary rel., port: type
    m : passive (2) % move. ports: id,id
    n : atomic (2) % nearest neighbour. ports: id,type
    n' : atomic (3) % near. neighb., answer. ports: id,id,type
    out : passive (0) % output, answer to latest query
    r : atomic (2) % range query. ports: id,id
    r' : atomic (3) % range query, answer. ports: id,id,dist
    right : atomic (1) % right part of binary rel., port: type
    s : passive (0) % searched
    tmp : passive (0) % for aux. nodes during 'near' search
    types : passive (0) % implements the subtyping relation
    void : atomic (1) % void, if we do not find any. ports: id
    w : atomic (1) % where. ports: id
    w' : atomic (2) % where, answer. ports: id,id
end

using Locmod

rule move =   % move a device, the sole reconfiguration query
  in(m_x,y) || c(d_x | [0]) || c_y([1]) || out([2])
    ->
  in() || c([0]) || c_y(d_x | [1]) || out()

rule where = % where is a certain device?
  in(w_x) || c_y(d_x | [0]) || out([1])
    ->
  in() || c_y(d_x | [0]) || out(w'_x,y)

% Invariant: Exactly one container is linked to name 'root'.
rule find_all_1 =       % find all devices (of type <=t), init
  in(f_t) || c_root(c([0])) || out([1])
    ->
  in(g) || c_root(c(f_t | s() | [0])) || out()
```

```
rule find_all_2 = % found dev of leq type, add to s and out
  c(f_t | [0] | d_x,t' | s([1])) ||
  types(leq(left_t' | right_t)) ||
  out([2])
    ->
  c(f_t | [0] | s(d_x,t' | [1])) ||
  types(leq(left_t' | right_t)) ||
  out(f'_x,t | [2])

rule find_all_3 = % found dev of gt type, add to s only
  c(f_t | [0] | d_x,t' | s([1])) ||
  types(gt(left_t' | right_t)) ||
  out([2])
    ->
  c(f_t | [0] | s(d_x,t' | [1])) ||
  types(gt(left_t' | right_t)) ||
  out([2])

rule find_all_4 = % no more here, go up while s-merging
  c(c(f_t | s([2])) | s([1]) | [0])
    ->
  c(f_t | s(c([2]) | [1]) | [0])

rule find_all_5 = % only containers here, go down
  c(f_t | c([1]) | s([2]) | [0])
    ->
  c(c(f_t | s() | [1]) | s([2]) | [0])

rule find_all_6 = % done, 'out' has all - clean up s,f,g
  in(g) || c_root(c(f_t | s([0])))
    ->
  in() || c_root([0])

rule range_dev_to_dev = % lookup range between two devices
  in(r_x,y) || c_i(d_x | [0]) || c_i(d_y | [1]) || out([2])
    ->
  in() || c_i(d_x | [0]) || c_i(d_y | [1]) || out(r'_x,y,i)

rule range_dev_to_con = % lookup range from device to leaf
```

```
  in(r_x,y) || c_i(d_x | [0]) || c_y,i([1]) || out([2])
    ->
  in() || c_i(d_x | [0]) || c_y,i([1]) || out(r_x,y,i)

rule near_1 = % find nearest neighbour of type <= t, init
  in(n_x,t) || c(d_x,t' | [0]) || out([1])
    ->
  in(g) || c(n_x,t | d_x,t' | s() | [0]) || out()

rule near_2 = % found one suitable in same container, done
  in(g) ||
  c(n_x,t | d_x,t' | d_y,t'' | s([1]) | [0]) ||
  types(leq(left_t'' | right_t)) ||
  out()
    ->
  in() ||
  c(d_x,t' | d_y,t'' | [1] | [0]) ||
  types(leq(left_t'' | right_t)) ||
  out(n'_x,y,t)

rule near_3 = % found one of wrong type, add to s and stay
  c(n_x,t | d_x,t' | d_y,t'' | s([1]) | [0]) ||
  types(gt(left_t'' | right_t))
    ->
  c(n_x,t | d_x,t' | s(d_y,t'' | [1]) | [0]) ||
  types(gt(left_t'' | right_t))

rule near_4 = % not here, go one up, create proxy d' & 'a'
  c(c(n_x,t | d_x,t' | s([1])) | [0]) || 1
    ->
  c(s(d_x,t' | [1]) | n_z,t | [0]) || tmp(d'_z | a)

rule near_5 = % only containers here, go down and create s
  c(n_x,t | c([1]) | [0]) || tmp(d'_x | a)
    ->
  c(c(n_x,t | s() | [1]) | [0]) || tmp(d'_x | a)

rule near_6 = % this c's children were empty, expand s, go up
  c(c(n_x,t | s([1])) | [0]) || tmp(d'_x | a)
    ->
```

```
  c(s(c([1])) | n_x,t | [0]) || tmp(d'_x | a)

rule near_7 = % found wrong type, enclose in s but stay
  c(n_x,t | d_y,t'' | s([1]) | [0])
  types(gt(left_t'' | right_t)) ||
  tmp(d'_x | a)
    ->
  c(n_x,t | s(d_y,t'' | [1]) | [0])
  types(gt(left_t'' | right_t)) ||
  tmp(d'_x | a)

% found suitable somewhere, replace 'a' with n' and
% disconnect n from d' while connecting d' with n'
rule near_8 =
  c(c(n_x,t | d_y,t'' | [1] | s([2])) | [0]) ||
  types(leq(left_t'' | right_t)) ||
  tmp(d'_x | a)
    ->
  c(s(c(d_y,t'' | [1] | [2])) | n_x,t | [0]) ||
  types(leq(left_t'' | right_t)) ||
  tmp(d'_p,u | n'_u,y,t)

rule near_9 = % searched all here, go up and cont. cleaning
  c(c(n_t | s([0])) | [1]) || tmp(d'_u | n'_u,t)
   ->
  c(s(c([0])) | n_t | [1]) || tmp(d'_u | n'_u,t)

rule near_10 = % enclose remaining c's before going up
  c(n_t | c([0]) | s([1]) | [2]) || tmp(d'_u | n'_u,t)
    ->
  c(n_t | s(c([0]) | [1]) | [2]) || tmp(d'_u | n'_u,t)

rule near_11 = % collapse s's to ensure all s's are cleaned up
  c(n | s([0]) | s([1]) | [2]) || tmp(d')
    ->
  c(n) | s([0] | [1]) | [2]) || tmp(d')

% search w. proxy reached root *without* finding one
rule near_12 =
  c_root(c(n_t | s([0]))) || tmp(d'_u | a)
```

```
    ->
  c_root(c([0])) || tmp(d'_u | n'_u,v,t | void_v | e)

rule near_13 = % search w. proxy reached root *with* 1 found
  c_root(c(n_t | s([0]))) || tmp(d'_u | n'_u,t)
    ->
  c_root(c([0])) || tmp(d'_u | n'_u,t | e)

rule near_14 = % return, when we had to use proxy
  c(d_x,t) || d_y || tmp(d'_x,u | n'_u,y,t | e) || in(g)
    ->
  c(d_x,t) || d_y || out(n'_x,y,t) || in()

rule near_15 = % return, special case
  c_root(c(d_x,t' | n_x,t)) || in(g) || 1
    ->
  c_root(c(d_x,t')) || out(n'_x,v,t | void_v) || in()

% can also find the distance to nearest d with type leq(t)
% by first calling 'near' and then just using dist-links
```

3.B.1 Concluding remarks on native queries

The main point to notice is that the "nearest neighbour" query takes 15 rules to be implemented. We do not believe that it can be done much simpler. The complexity of even this conceptually simple query is close to the limit of what we can claim to understand completely. For more complicated programming tasks we will likely run into trouble trying to program directly with bigraphical reaction rules.

3.C Rigid control-sortings and RPOs

This section is due to Søren Debois.

For a bigraph b (sorted or otherwise), we write b^* for the function that takes each place (site or root) or node of b to its uniquely determined root. In this appendix we will generally omit writing down the link-graph part of interfaces when we do not need them.

Definition 3.11 (Rigid control-sorting)**.** *Let $\mathcal{K}$ be a set of controls. A sorting $\mathcal{S} = (\mathcal{K}, \Theta, \Phi)$ is a* rigid control-sorting *if $\Theta \subseteq \mathcal{P}(\mathcal{K})$ and there exists a predicate ϕ,*

such that

$$\Phi\Big((m,s_m) \xrightarrow{f} (n,s_n)\Big) \quad \textit{iff} \quad \begin{cases} (i) & s_m(i) = s_n({}^*f(i)) & \textit{for } i < m, \\ (ii) & \phi(ctrl_f(v), s_n(f^*(v))) & \textit{for } v \textit{ node in } f. \end{cases}$$

In the sequel, we assume a fixed set of controls $\mathcal{K}$, rigid control-sorting $\mathcal{S} = (\mathcal{K}, \Theta, \Phi)$, a sorted signature $\Sigma^{\mathcal{S}}$ and a corresponding unsorted signature $\mathcal{U}(\Sigma^{\mathcal{S}}) = \Sigma$; following [Jen07], we write $\mathbf{Big}(\Sigma)$ for the precategory of concrete bigraphs over Σ and $\mathbf{Big}(\Sigma^{\mathcal{S}})$ for the corresponding precategory of sorted concrete bigraphs, and we write $\mathcal{U}$ for the forgetful functor from $\mathbf{Big}(\Sigma^{\mathcal{S}})$ to $\mathbf{Big}(\Sigma)$; recall that this functor is faithful.

In Theorem 3.14 we state that $\mathbf{Big}(\Sigma^{\mathcal{S}})$ has RPOs; it follows that the standard bisimulation on $\mathbf{Big}(\Sigma^{\mathcal{S}})$ is a congruence. To establish Theorem 3.14, we will need some lemmas to make precise just how closely $\mathbf{Big}(\Sigma^{\mathcal{S}})$ mimics $\mathbf{Big}(\Sigma)$.

Lemma 3.12. *If $\mathcal{U}(a) = p \circ q$, then there exists unique b, c s.t. $\mathcal{U}(b) = p$, $\mathcal{U}(c) = q$ and $a = b \circ c$.*

Proof. For existence, suppose $a : (m, s_m) \longrightarrow (n, s_n)$ and $cod(q) = dom(p) = l$. Define

$$s_l(i) \stackrel{\text{def}}{=} s_n(p^*(i)). \tag{3.11}$$

We claim that $c = (m, s_m) \longrightarrow q(l, s_l)$ and $b = (l, s_l) \longrightarrow p(n, s_n)$ are well-sorted. Consider c. Condition (i) of Definition 3.11 is satisfied by (3.11), Condition (ii) is satisfied because the nodes of c is a subset of the nodes of a. Now consider b. For $i < n$, we find

$$s_m(i) = s_n({}^*a(i)) = s_n({}^*p({}^*q(i))) = s_l({}^*q(i)),$$

satisfying Condition (i). Next, for v a node of q, we find

$$\phi(ctrl_q(v), s_l(q^*(v))) = \phi(ctrl_a(v), s_n({}^*p({}^*q(v)))) = \phi(ctrl_a(v), s_n({}^*a(v))).$$

But $\phi(ctrl_a(v), s_n(a^*(v)))$ is satisfied by well-sortedness of a; thus Condition (ii) is satisfied.

For uniqueness, it is sufficient to prove that s_l is the only sorting making b and c well-sorted. Suppose s'_l is an alternate such sorting. If there is $i < l$ s.t. $s'_l(i) \neq s_l(i) = s_n(p^*(i))$, then $(l, s'_l) \xrightarrow{p} (n, s_n)$ is not well-sorted: contradiction. Thus $s'_l = s_l$. □

Lemma 3.13. *If a, b is a cospan and $\mathcal{U}(a) = \mathcal{U}(b)$, then $a = b$.*

Proof. Because $\mathcal{U}(a) = \mathcal{U}(b)$, a and b must have the same inner width, m:

$$(m, s_m) \xrightarrow{\quad a \quad} (n, s_n) \xleftarrow{\quad b \quad} (m, s'_m).$$

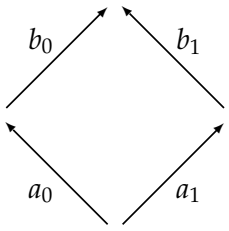
Suppose for a contradiction that there is $i < m$ s.t. $s'_m(i) \neq s_m(i)$. Then

$$s_n(a^*(i)) = s_m(i) \neq s'_m(i) = s_n({}^*b(i)),$$

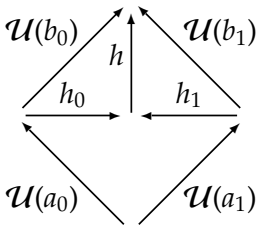
but that cannot be, because $a^*(i) = {}^*b(i)$ follows from $\mathcal{U}(a) = \mathcal{U}(b)$: contradiction. □

Theorem 3.14. $\mathbf{Big}(\Sigma^{\mathcal{S}})$ *has RPOs.*

Proof. Consider the square (i) below.

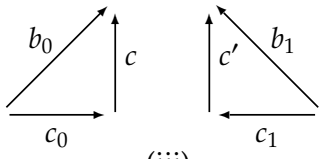


(i)



(ii)

Apply $\mathcal{U}$ to get a similar square in $\mathbf{Big}(\Sigma)$, and erect an RPO there, altogether obtaining the diagram (ii). By Lemma 3.12, there are c_0 and c factoring b_0 s.t. $\mathcal{U}(c_0) = h_0$ and $\mathcal{U}(c) = h$; symmetrically, there are also c_1 and c' factoring b_1 s.t. $\mathcal{U}(c_1) = h_1$ and $\mathcal{U}(c') = h$. (See diagram (iii) below.)



(iii)

But b_0, b_1 is a cospan, so also c, c' is a cospan; thus $c = c'$ by Lemma 3.13, and we have a candidate RPO c_0, c_1, c.

Suppose d_0, d_1, d is an alternate candidate RPO. We must find unique e s.t. $c = d \circ e$. Because $\mathcal{U}(c_0), \mathcal{U}(c_1), \mathcal{U}(c)$ is an RPO, we find unique p s.t. $\mathcal{U}(c) = \mathcal{U}(d) \circ p$. By Lemma 3.12, there are unique d', e s.t. $\mathcal{U}(d') = \mathcal{U}(d)$, $\mathcal{U}(e) = p$ and $c = d' \circ e$. But then d, d' is cospan, so by Lemma 3.13, $d = d'$.

Thus, we have found e s.t. $c = d \circ e$. For uniqueness, suppose there is e' with $c = d \circ e'$. Then

$$h = \mathcal{U}(c) = \mathcal{U}(d) \circ \mathcal{U}(e') = \mathcal{U}(d) \circ p$$

but then $\mathcal{U}(e') = p = \mathcal{U}(e)$ by uniqueness of p; but e', e is also a cospan, so by Lemma 3.13, $e = e'$. □

Corollary 3.15. *Bisimulation on the standard transition-system of* $\mathfrak{Big}(\Sigma^{\mathcal{S}})$ *is a congruence.*

Proof. By [Jen07, Theorem 3.16], possession of RPOs is a sufficient prerequisite for the desiderata. □

We can now prove that the sorting of Definition 3.8 gives a congruential bisimulation.

Theorem 3.16. *Let* $\mathcal{S}$ *be the sorting given in Definition 3.8. Then the bisimulation over the standard transitions of* $\mathfrak{Big}(\Sigma^{\mathcal{S}})$ *is a congruence.*

Proof. By Corollary 3.15, it is sufficient to show that $\mathcal{S}$ is a rigid control sorting. Take $\phi(k, K) = k \in K$. Clearly, $\Phi\Big((m, s_m) \xrightarrow{f} (n, s_n)\Big)$ is equivalent to $i < m \implies s_m(i) = s_n({}^*f(i))$ and $v \in f \implies \phi(ctrl_f(v), s_n(f^*(v)))$. □

We have developed Plato-graphical models in this chapter. In the next chapter we continue with another theoretical development, namely the representation of the MiniML programming language with references in Bigraphs, which we shall need for our bigraphical model in Chapter 5.

4

Encoding MiniML with References in Bigraphs

4.1 Purpose

In this chapter we encode a MiniML-like calculus with references, Ξ, in Bigraphs. The motivation is two-fold: (1) We would like to be able to express some parts of our bigraphical location model (see Chapter 5) in Ξ using the Plato-graphical ability to combine several languages. (2) It is an interesting study in itself to investigate how one may model references in Bigraphs, because to the best of our knowledge no previous encodings of calculi with side effects have been studied in Bigraphs.

Recall that we found it useful to be able to express some parts of our system in a high-level language, Chapter 3. We claim that Ξ is such a language. We need references for the location model part **L** and the agent part **A** of our location model in Chapter 5. For now, we ask the reader to trust us when we say that references are needed for encapsulating the state of the location model with respect to the agent and the sensor.

We investigate the use of closed links in Local Bigraphs as defined in [Mil04c, Mil07]. This chapter aims to

- further investigate a question of Chapter 3, namely which high-level languages that can be encoded in Bigraphs for use in Plato-graphical models,
- communicate a few humble insights about the behaviour of closed links (edges) to readers with some knowledge of Bigraphs,
- to experiment with and illustrate some non-trivial *matches*,
- and to show how references can be encoded in (Local) Bigraphs using closed links.

First, we find that closed links can not interfere with outer names (open links) or with each other, and that they behave as they do in Binding Bigraphs when they are replicated as part of a parameter. Second, we find that a calculus with references can be encoded in (Local) Bigraphs using closed links. There are some subtleties associated with the encoding and the resulting reaction relation, which we address as we go along.

4.2 Non-interference of closed links

Recall that the binding *arity* of a control $\mathsf{K} : b \to f$ is a pair of finite ordinals, b is the *binding arity* and f the *free arity*. Consider the following binding signature Σ with a parametric reaction rule.

Control	Activity	Arity	Comment
loca	active	$0 \to 1$	Nested location (e.g. a room)
devi	atomic	$0 \to 1$	Mobile device

$$\mathsf{loca}_g(-_0) \rightarrowtriangle /x.\mathsf{loca}_g(\mathsf{devi}_x \mid -_0) \tag{4.1}$$

Notation 4.1. *We introduce the* operator $/x.B$ *as a generalisation of* $/x \circ B$*, in the sense that* $/x.B$ *works for all widths of* B*, and it marks where identities are implicit in a composition. It binds less than* $|$*,* $\|$*, and* $/$*.* $\mid$ *binds tighter than* $\|$ *which binds tighter than* $/$*.*

By Definition 3.2 (parametric reaction rule) of [Mil04c] and the additional clarification in Section 3 of [Mil07], our parametric reaction rule (4.1) is of the form

$$(R : I \to K, R' : I' \to K, \eta, \vec{\iota})$$

where $I = \vec{X}$ and $I' = \vec{X}'$ are partitions (i.e. disjoint) with widths m and m', and $\eta : m' \to m$ is a map of ordinals. The fourth component is a vector of bijections $\vec{\iota}_j : X_{\eta(j)} \to X'_j$ for each $j \in m'$.

A parametric rule generates ground rules of the form

$$((R \oplus \omega) \circ a, (R' \oplus \omega') \circ a')$$

where $I \oplus H$, $I' \oplus H'$ and $K \oplus L$ are interface extensions with $H' = \overline{\eta}(H)$[1]. Let $\omega : H \to L$ and $\omega' : H' \to L$ be wirings that agree on the names of H' and have the same support, i.e., $|\omega| = |\omega'|$. Then for any $a : I \oplus H$, complete the ground rule by defining $a' = \overline{\eta}_{\vec{\iota}}(a) : I' \oplus H'$.

[1] $\overline{\eta}$ is the *instantiation* map.

Notation 4.2. *We follow the short-hand notation of [Mil04c] and write a local interface as a vector of names. We omit the parentheses if the vector is of size one, and the curly brackets if the set is a singleton. We omit the empty inner face ϵ of ground bigraphs.*

The components of rule (4.1) are as follows:

$$\begin{aligned} R &= \mathsf{loca}_g(-_0) : \emptyset \to g \\ R' &= /x.\mathsf{loca}_g(\mathsf{devi}_x \mid -_0) : \emptyset \to g \\ \eta &= \mathsf{Id}_1 \\ \vec{\iota} &= (\mathsf{Id}_\emptyset) \end{aligned}$$

where Id_1 is the identity function on the finite ordinal $1 = \{0\}$, and $\vec{\iota}$ is a vector of isomorphisms relating located names in the parameters (none in this case). We see that $x \neq g$ since otherwise $cod(R) \neq cod(R')$. Thus, we could just as well have chosen $R' = \mathsf{loca}_g(/x.\mathsf{devi}_x \mid -_0)$, since these two terms denote the same bigraph, which is apparent graphically, and justified by Proposition 2.7 (open decomposition) of [Mil04c]. In the term language, however, one must be careful when choosing names; if $x = g$, then $\mathsf{loca}_g(/x.\mathsf{devi}_x \mid -_0) \neq /x.\mathsf{loca}_g(\mathsf{devi}_x \mid -_0) = (\mathsf{id}_g \oplus /x) \circ \mathsf{loca}_g(\mathsf{devi}_x \mid -_0)$ because $\mathsf{id}_g \oplus /x$ is not defined when $g = x$. We see that when writing reaction rules we need not worry about a closed link "capturing" outer names by accident during reaction. This is how it should be.

Now consider, informally, the following situation: What happens if we apply rule (4.1) twice consecutively? If we think of a device's outer name as its identifier, can we accidentally reuse (the name of) a closed link and thereby identify two different devices? That is, can we formally have the following sequence of reactions:

$$\mathsf{loca}_p() \twoheadrightarrow /f.\mathsf{loca}_p(\mathsf{devi}_f) \twoheadrightarrow /f.\mathsf{loca}_p(\mathsf{devi}_f \mid \mathsf{devi}_f) \ ?$$

The answer is no; the second reaction is not allowed. Let us convince ourselves by looking at an example.

Consider a bigraph $B_1 = \mathsf{loca}_p()$, then $B_1 \twoheadrightarrow B_2$ if and only if there exist $C, \omega, a, \omega', a'$ such that

$$\begin{aligned} B_1 &= C \circ (R \oplus \omega) \circ a \\ B_2 &= C \circ (R' \oplus \omega') \circ a' \end{aligned}$$

where ω is a wiring and parameter a is discrete (for $\emptyset$ and g). Clearly, we

must have

$$\begin{aligned} a &= 1 : \emptyset \\ \omega &= \mathsf{id}_\emptyset : \emptyset \to \emptyset \\ C &= p/g : g \to p \\ a' &= a \\ \omega' &= \omega \end{aligned}$$

where $B_2 = /f.\mathsf{loca}_p(\mathsf{devi}_f) = \mathsf{loca}_p(/f.\mathsf{devi}_f)$ for any $f \neq p$ (forced by the rule). Clearly, the choice of the name f is insignificant, because it does not appear in the outer (or inner) face of B_2 when closed, and since edges (closed links) do not have identity in abstract bigraphs. Thus, $/f.\mathsf{loca}_p(\mathsf{devi}_f) = /k.\mathsf{loca}_p(\mathsf{devi}_k)$ for any $f, k \neq p$.

We wish to apply (4.1) to B_2. B_2 can be matched by (4.1) in (at least) two ways depending on whether the closure of f is done in the wirings ω, ω' or in the context C. If we close f in the context then the wirings ω, ω' will be *placings*, and Proposition 3.5 (placings suffice) of [Mil04c] states that it suffices to consider such ground rules. To accentuate this we write π, π' for the wirings (placings), and we have

$$\begin{aligned} a &= \mathsf{devi}_f : f \\ \pi &= \mathsf{id}_f : f \to f \\ C &= p/g \oplus /f : \{g, f\} \to p \\ a' &= a \\ \pi' &= \pi \end{aligned}$$

where $B_3 = /h./f.\mathsf{loca}_p(\mathsf{devi}_h \mid \mathsf{devi}_f)$ for arbitrary $f, h \neq p$. Graphically, it is clear that the order of closures is insignificant so this also holds in the term language. Could we have chosen $f = h$? No, because that would render C undefined. We conclude that closed links can not interfere with one another, which is apparent in the graphical representation.

Now, let us consider replication of parameters involving closed links in Local Bigraphs. To illustrate, we consider the "replication rule" of [JM04] (p. 78):

$$\begin{aligned} R &= \mathsf{rep}(-_0) : \emptyset \to \emptyset \\ R' &= -_0 \mid \mathsf{rep}(-_1) : (\emptyset, \emptyset) \to \emptyset \\ \eta &= \{0, 1 \mapsto 0\} \\ \vec{\iota} &= (\mathsf{Id}_\emptyset, \mathsf{Id}_\emptyset) \end{aligned}$$

Notice how this rule does not say anything about linking. Applying this rule to the bigraph $B = /l.\mathsf{rep}(\mathsf{u}_l \mid \mathsf{v}_l)$ we obtain $B' = /l.(\mathsf{u}_l \mid \mathsf{v}_l) \mid \mathsf{rep}(\mathsf{u}_l \mid \mathsf{v}_l)$ because

$$\begin{aligned} a &= \mathsf{u}_x \mid \mathsf{v}_y : \{x, y\} \\ \pi &= \mathsf{id}_{\{x,y\}} : \{x, y\} \to \{x, y\} \\ C &= /l.(l/x \mid l/y) : \{x, y\} \to \emptyset \\ a' &= (\mathsf{u}_x \mid \mathsf{v}_y) \parallel (\mathsf{u}_x \mid \mathsf{v}_y) \\ \pi' &= \mathsf{id}_{\{x,y\}} \mid \mathsf{id}_{\{x,y\}} : (\{x, y\}, \{x, y\}) \to \{x, y\} \end{aligned}$$

The closure resides in C so the global outer names of the two replicas are identified (must point to the same edge). If one wishes each replica to have its own private link, one should use binding ports or replicate using, e.g., the following rule, which is a one-time replication rule for simplicity.

$$\begin{aligned} R &= /x.\mathsf{rep}(-_0\langle x\rangle) : \{x\} \to \emptyset \\ R' &= /x.-_0\langle x\rangle \mid /y.-_1\langle y\rangle : (x, y) \to \emptyset \\ \eta &= \{0, 1 \mapsto 0\} \\ \vec{\iota} &= (x \mapsto x, x \mapsto y) \end{aligned}$$

With this rule we have $B = /l.\mathsf{rep}(\mathsf{u}_l \mid \mathsf{v}_l) \twoheadrightarrow B' = (/l.\mathsf{u}_l \mid \mathsf{v}_l) \mid (/l.\mathsf{u}_l \mid \mathsf{v}_l)$. We show the components of the match to illustrate the "trick" of closing the links in the rule so that the context can not identify outer names before closing them. The intuition is that the context C has no handle on the links and thus can not manipulate them; x is closed in R so u and v in a can not have different outer names because C can not reunite them.

$$\begin{aligned} a &= \mathsf{u}_x \mid \mathsf{v}_x : x \\ \omega &= \mathsf{id}_\emptyset : \emptyset \to \emptyset \\ C &= \mathsf{id}_\emptyset : \emptyset \to \emptyset \\ a' &= \mathsf{u}_x \mid \mathsf{v}_x \parallel \mathsf{u}_i \mid \mathsf{v}_j : (x, y) \\ \omega' &= \mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset : (\emptyset, \emptyset) \to \emptyset \end{aligned}$$

This concludes our treatment of closed link non-interference.

4.3 Encoding references via closed links

Consider the following MiniML-like call-by-value calculus Ξ with pairs and projections, references, datatype constructors and deconstructors, fixed-points, and natural numbers. We need some notation.

Notation 4.3. *Denote the dereferencing operation by* ! *(bang). n ranges over the set $\mathcal{N}$ of natural numbers (including zero). x, f range over an infinite set $\mathcal{V}$ of variable names with members x, y, z and so forth. l ranges over a set $\mathcal{L}$ of* reference cells. D *ranges over an infinite set $\mathcal{D}$ of data type names.* C *ranges over an infinite set $\mathcal{C}$ of constructor names with members* C0, C1, C2 *and so forth. The sets $\mathcal{V}$, $\mathcal{L}$, $\mathcal{D}$, and $\mathcal{C}$ are pairwise disjoint. We use the shorthand notation '*case e of $\mathsf{C}_i\ x_i \Rightarrow e_i\ ^{i=0..n}$*' for '*case e of $\mathsf{C}_0\ x_0 \Rightarrow e_0 \mid \mathsf{C}_1\ x_1 \Rightarrow e_1 \mid \ldots \mid \mathsf{C}_n\ x_n \Rightarrow e_n$*'. Likewise for data-type declarations.*

Here is the calculus Ξ in form of a BNF grammar with programs p, terms e, values v, and evaluation contexts E. Notice that we allow base type declarations t_i when declaring new data types. The reason is that we want to use the SML runtime system in the implementation.

Definition 4.4 (Syntax)**.**

$$
\begin{array}{rcl}
p & ::= & \mathsf{datatype}\ \mathsf{D} = \mathsf{C}_i\ \mathsf{of}\ t_i\ ^{i=0..n}; p \mid e \\
e & ::= & x \mid e_1\ e_2 \mid (e_1, e_2) \mid \mathsf{fst}\ e \mid \mathsf{snd}\ e \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2\ \mathsf{end} \mid \\
 & & \mathsf{ref}\ e \mid\ !e \mid e_1 := e_2 \mid \mathsf{C}\ e \mid \mathsf{case}\ e\ \mathsf{of}\ \mathsf{C}_i\ x_i \Rightarrow e_i\ ^{i=0..n} \mid v \\
v & ::= & \lambda x.e \mid \mathsf{fix}\ f(x) = e \mid (v_1, v_2) \mid \mathsf{unit} \mid l \mid \mathsf{C}\ v \mid n \\
E & ::= & [\,] \mid (E, e) \mid (v, E) \mid \mathsf{fst}\ E \mid \mathsf{snd}\ E \mid \mathsf{let}\ x = E\ \mathsf{in}\ e\ \mathsf{end} \mid \\
 & & \mathsf{let}\ x = v\ \mathsf{in}\ E\ \mathsf{end} \mid E\ e \mid v\ E \mid \mathsf{ref}\ E \mid\ !E \mid E := e \mid v := E \mid \\
 & & \mathsf{C}\ E \mid \mathsf{case}\ E\ \mathsf{of}\ \mathsf{C}_i\ x_i \Rightarrow e_i\ ^{i=0..n}
\end{array}
$$

A program is a possibly empty sequence of data-type declarations followed by an expression. Concrete cell constants (ranged over by l) only arise in terms that are the intermediate results of evaluation; they are not in the language in which programmers write. The evaluation strategy is call-by-value (CBV). We have chosen to explicitly include several constructs in the calculus that could have been encoded instead. This is done to avoid cluttering the presentation and use of the calculus with encodings. Should we, however, wish to formally prove properties about this calculus then encodings would be preferred to limit the number of cases in inductive analyses.

We introduce a *store* to keep track of *store cell* values, and define dynamics via a single-step evaluation relation $\longrightarrow$ on *configurations*. We use the following notational conventions.

Notation 4.5. *σ ranges over stores, i.e., partial functions from locations to values, $(\sigma, l \mapsto v)$ denotes binding of location l to value v, $\sigma[l \mapsto v]$ denotes updating of*

store σ with location l now bound to value v, and $\{v/x\}e$ is the result of replacing all free occurrences of variable x in expression e by value v.

Before defining the semantics we need to define substitution of values in place of variables in terms.

Definition 4.6 (Substitution)**.**

$$
\begin{array}{rcl}
\{v/x\}x & \stackrel{\text{def}}{=} & v \\
\{v/x\}y & \stackrel{\text{def}}{=} & y \\
\{v/x\}(e_1 e_2) & \stackrel{\text{def}}{=} & (\{v/x\}e_1)(\{v/x\}e_2) \\
\{v/x\}(\text{fst } e) & \stackrel{\text{def}}{=} & \text{fst } (\{v/x\}e) \\
\{v/x\}(\text{snd } e) & \stackrel{\text{def}}{=} & \text{snd } (\{v/x\}e) \\
\{v/x\}(\text{let } y = e_1 \text{ in } e_2 \text{ end}) & \stackrel{\text{def}}{=} & \text{let } y = \{v/x\}e_1 \text{ in } \{v/x\}e_2 \text{ end} \quad , \\
 & & \textit{if } y \neq x \\
\{v/x\}(\text{ref } e) & \stackrel{\text{def}}{=} & \text{ref } \{v/x\}e \\
\{v/x\}(!e) & \stackrel{\text{def}}{=} & !\{v/x\}e \\
\{v/x\}(e_1 := e_2) & \stackrel{\text{def}}{=} & \{v/x\}e_1 := \{v/x\}e_2 \\
\{v/x\}(\text{C } e) & \stackrel{\text{def}}{=} & \text{C } (\{v/x\}e) \\
\{v/x\}(\text{case } \text{C}_j v' \text{ of } \text{C}_i\ x_i \Rightarrow e_i^{\,i=0..n}) & \stackrel{\text{def}}{=} & \text{case } \text{C}_j v' \text{ of } \text{C}_i\ x_i \Rightarrow (\{v/x\}e_i)^{\;i=0..n} \\
\{v/x\}(\lambda y.e) & \stackrel{\text{def}}{=} & \lambda y.\{v/x\}e \quad , \quad \textit{if } y \neq x \\
\{v/x\}(\text{fix } f(y) = e) & \stackrel{\text{def}}{=} & \text{fix } f(y) = \{v/x\}e \quad , \quad \textit{if } f, y \neq x \\
\{v/x\}\text{unit} & \stackrel{\text{def}}{=} & \text{unit} \\
\{v/x\}l & \stackrel{\text{def}}{=} & l \\
\{v/x\}n & \stackrel{\text{def}}{=} & n
\end{array}
$$

Notice that in the case for lambda abstractions and fixed-point constructs there is no need for a side-condition stating that $y \notin \mathsf{fv}(v)$, where function fv returns the free variables of a term, because only values v, which are closed terms by definition, are inserted.

We call the following rules *basic*.

Definition 4.7 (Semantics)**.**

$$
\begin{array}{rcll}
\langle \text{fst}\ (v_1, v_2), \sigma\rangle & \longrightarrow & \langle v_1, \sigma\rangle & \\
\langle \text{snd}\ (v_1, v_2), \sigma\rangle & \longrightarrow & \langle v_2, \sigma\rangle & \\
\langle \text{let}\ x = v\ \text{in}\ e\ \text{end}, \sigma\rangle & \longrightarrow & \langle \{v/x\}e, \sigma\rangle & \\
\langle \text{ref}\ v, \sigma\rangle & \longrightarrow & \langle l, (\sigma, l \mapsto v)\rangle & , \quad l \in \mathcal{L}\ fresh \\
\langle !l, \sigma[l \mapsto v]\rangle & \longrightarrow & \langle v, \sigma[l \mapsto v]\rangle & \\
\langle l := v', \sigma[l \mapsto v]\rangle & \longrightarrow & \langle \text{unit}, \sigma[l \mapsto v']\rangle & \\
\langle (\lambda x.e)\ v, \sigma\rangle & \longrightarrow & \langle \{v/x\}e, \sigma\rangle & \\
\langle (\text{fix}\ f(x) = e)\ v, \sigma\rangle & \longrightarrow & \langle \{v/x, (\text{fix}\ f(x) = e)/f\}e, \sigma\rangle & \\
\langle \text{case}\ C_j\ v\ \text{of}\ C_i\ x_i \Rightarrow e_i\ ^{i=0..n}, \sigma\rangle & \longrightarrow & \langle \{v/x_j\}e_j, \sigma\rangle & , \quad \text{if } j \in \{0, \ldots, n\}
\end{array}
$$

Notice that stores are *not* terms. We briefly explain the last rule. It allows evaluation of a case construct provided that the expression to be matched is a value, and that it matches one of the constructors declared. The result of the evaluation is a substitution of v for x_j in the j^{th} branch of the case construct. There is only one variable on the left-hand side in each branch, which means that if one wishes to match a constructor with a value that is, e.g., a pair then the variable x_j has to be manually deconstructed (in this case projected) on the right-hand side of the matching branch. We will see an example of this in Chapter 5. We close evaluation under contexts:

Definition 4.8. *If $\langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle$ then there exists a unique E such that $e = E[r]$, $\langle r, \sigma\rangle \longrightarrow \langle r', \sigma'\rangle$ is a reduction by one of the* basic *rules, and $e' = E[r']$.*

We encode Ξ in Local Bigraphs [Mil07]. The signature is shown in Figure 4.1 to be explained shortly.

For now, let $\mathsf{i0}, \mathsf{i1}, \mathsf{i2}, \ldots$ represent natural numbers. These are used instead of a more cumbersome bigraphical representation, which will be presented in Section 4.3.1, of natural numbers using only constants *zero* and *successor*. Binding ports are used to delimit variable scope (see e.g. letb, lam, and casee). The controls letb, casee, lam, and fix, are passive to prevent evaluation "under them". The val control explicitly marks values such that we know when to lift the exp controls delaying evaluation. val is active to allow for substitutions to move "under it", a point to which we will return shortly. The last six controls are not images of Ξ-terms, but are introduced to implement a store, call-by-value (CBV) evaluation, and substitution. cell denotes a reference cell generated by evaluation. cell′ denotes a store cell, i.e., it resides in store, holding the value of a reference cell, which it refers

Control	Activity	Arity	Comment
var	atomic	$0 \to 1$	Variable
app	active	$0 \to 0$	Application
appl	active	$0 \to 0$	Left part of application
appr	active	$0 \to 0$	Right part of application
pair	active	$0 \to 0$	Pair
pairl	active	$0 \to 0$	Left component of pair
pairr	active	$0 \to 0$	Right component of pair
fst	active	$0 \to 0$	Left projection on pair
snd	active	$0 \to 0$	Right projection on pair
let	active	$0 \to 0$	Let construction
letd	active	$0 \to 0$	Declaration of 'let'
letb	passive	$1 \to 0$	Body of 'let'
ref	active	$0 \to 0$	Reference request
deref	active	$0 \to 0$	Dereference
assign	active	$0 \to 0$	Assignment
acell	active	$0 \to 0$	Cell component of assignment
aval	active	$0 \to 0$	Value component of assignment
case	active	$0 \to 0$	Case construct
casel	active	$0 \to 0$	Constructor to be matched of 'case'
casee	passive	$1 \to 0$	Case branch of 'case'
Cn	active	$0 \to 0$	A control for each member of C
val	active	$0 \to 0$	Value
lam	passive	$1 \to 0$	Lambda abstraction
fix	passive	$2 \to 0$	Fixed-point construction
unit	atomic	$0 \to 0$	Unit, for side effects
i0, i1, i2 ...	atomic	$0 \to 0$	Natural numbers
cell	atomic	$0 \to 1$	Generated reference cell (term)
cell'	active	$0 \to 1$	Generated store cell
store	active	$0 \to 0$	Store
exp	passive	$0 \to 0$	Delay evaluation
sub	active	$1 \to 0$	Term to undergo substitution
def	active	$0 \to 1$	Term to be inserted

Figure 4.1: Signature Σ_{Ξ}.

to by a closed link. The store and its cells are active so that the contents of the store and its cells can be updated, but no computation takes place in the store or the cells, because only values are stored there. As will become apparent when translating Ξ-terms into Local Bigraphs, exp is used to delay evaluation certain places in terms to implement the CBV semantics. sub and def are used to perform explicit substitution, sub holds the term to undergo substitution for some x and def holds the value to be inserted instead of x. The purposes of the other controls should be clear.

In Figure 4.2 we provide a formal translation of Ξ into Local Bigraphs assuming well-formed Ξ-programs. The semantic function $(\!|\cdot|\!)$ translates programs p, and the semantic function $[\![\cdot]\!]$ translates expressions e. The idea is to preserve the call-by-value semantics by inserting passive exp controls to hinder premature evaluation. Binding ports are used to limit the scope of variables, like lambda abstractions in the λ-calculus.

Notation 4.9. *We write $f \circ g$ as fg and* datatype *as* dt. *Composition binds tighter than prime product (on the right-hand sides of Figure 4.2).*

Consider Figure 4.2. The subscript set X on the translation *includes* the names of all free variables (fv) in e. Thus, each term e has many bigraph images. All the images of $[\![\cdot]\!]_X$ are ground so they all have the empty inner face ϵ, and the subscript set as outer face. We require λ-terms to be α-converted in such a way that the binding variables are all different. α-convertible Ξ-terms have equal images, e.g. $[\![\lambda y.y]\!]_\emptyset = [\![\lambda z.z]\!]_\emptyset$. Notice how we do *not* translate configurations, merely terms. This is because the store is implicit in the Ξ-world, but we need to model it explicitly in Bigraphs. The store is assumed to be empty at the beginning of evaluation. The sites $-_n$ (for $n \in \mathcal{N}$) are really identity morphisms.

The parametric bigraphical reaction rules corresponding to the dynamic single-step semantics of Ξ are presented in Figure 4.3. There is a dynamic correspondence between $\longrightarrow$ and $\rightarrowtriangle$, namely that $\longrightarrow$ can be mimicked by one or more uses of $\rightarrowtriangle$.

During evaluation some variables and cells may disappear and new cells may be created. We demand that $\mathsf{fv}(e) \subseteq X$, which is an invariant.

The first rule we wish to emphasise is rule (4.6). Evaluation of a 'let' expression results in a substitution. We emphasise the fact that $-_0$ in the redex can have x in its outer face and maintain it in the reactum as explained earlier in this chapter.

Rule (4.7) is wide because we wish to have the store at "top level" and not embedded in the program as such. The link is closed outermost

$$
\begin{array}{rcl}
(\!|\text{dt D} = \text{C}_i \text{ of } t_i \ {}^{i=0..n}; p|\!)_X & = & (\!|p|\!)_X \\
(\!|\text{dt D} = \text{C}_i \text{ of } t_i \ {}^{i=0..n}; e|\!)_X & = & [\![e]\!]_X \mid \mathsf{store}() \\
[\![x]\!]_{X \uplus \{x\}} & = & X \oplus \mathsf{var}_x \\
[\![(e_1, e_2)]\!]_X & = & (\mathsf{pair} \oplus \mathsf{id}_X) \\
 & & \Big((\mathsf{pairl} \oplus \mathsf{id}_X)[\![e_1]\!]_X \mid \\
 & & \quad (\mathsf{pairr} \oplus \mathsf{id}_X)(\mathsf{exp} \oplus \mathsf{id}_X)[\![e_2]\!]_X\Big) \\
[\![\text{fst } e]\!]_X & = & (\mathsf{fst} \oplus \mathsf{id}_X)[\![e]\!]_X \\
[\![\text{snd } e]\!]_X & = & (\mathsf{snd} \oplus \mathsf{id}_X)[\![e]\!]_X \\
[\![\text{let } x = e_1 \text{ in } e_2 \text{ end}]\!]_X & = & (\mathsf{let} \oplus \mathsf{id}_X) \\
 & & \Big((\mathsf{letd} \oplus \mathsf{id}_X)[\![e_1]\!]_X \mid \\
 & & \quad (\mathsf{letb}_{(x)} \oplus \mathsf{id}_X)[\![e_2]\!]_{X \uplus \{x\}}\Big) \\
[\![\lambda x.e]\!]_X & = & (\mathsf{val} \oplus \mathsf{id}_X)(\mathsf{lam}_{(x)} \oplus \mathsf{id}_X)[\![e]\!]_{X \uplus \{x\}} \\
[\![\text{fix } f(x) = e]\!]_X & = & (\mathsf{val} \oplus \mathsf{id}_X)(\mathsf{fix}_{(f,x)} \oplus \mathsf{id}_X)[\![e]\!]_{X \uplus \{f,x\}} \\
[\![e_1\ e_2]\!]_X & = & (\mathsf{app} \oplus \mathsf{id}_X) \\
 & & \Big((\mathsf{appl} \oplus \mathsf{id}_X)[\![e_1]\!]_X \mid \\
 & & \quad (\mathsf{appr} \oplus \mathsf{id}_X)(\mathsf{exp} \oplus \mathsf{id}_X)[\![e_2]\!]_X\Big) \\
[\![\text{ref } e]\!]_X & = & (\mathsf{ref} \oplus \mathsf{id}_X)[\![e]\!]_X \\
[\![!e]\!]_X & = & (\mathsf{deref} \oplus \mathsf{id}_X)[\![e]\!]_X \\
[\![e_1 := e_2]\!]_X & = & (\mathsf{assign} \oplus \mathsf{id}_X) \\
 & & \Big((\mathsf{acell} \oplus \mathsf{id}_X)[\![e_1]\!]_X \mid \\
 & & \quad (\mathsf{aval} \oplus \mathsf{id}_X)(\mathsf{exp} \oplus \mathsf{id}_X)[\![e_2]\!]_X\Big) \\
[\![\text{C } e]\!]_X & = & (\mathsf{C} \oplus \mathsf{id}_X)[\![e]\!]_X \\
[\![\text{case } e \text{ of } C_i\ x_i \Rightarrow e_i \ {}^{i=0..n}]\!]_X & = & (\mathsf{case} \oplus \mathsf{id}_X) \\
 & & \Big((\mathsf{casel} \oplus \mathsf{id}_X)[\![e]\!]_X \mid \\
 & & \quad (\mathsf{casee}_{(x_0)} \oplus \mathsf{id}_X)(\mathsf{C}0 \oplus \mathsf{id}_X)[\![e_0]\!]_{X \uplus x_0} \\
 & & \quad \vdots \\
 & & \quad (\mathsf{casee}_{(x_n)} \oplus \mathsf{id}_X)(\mathsf{C}n \oplus \mathsf{id}_X)[\![e_n]\!]_{X \uplus x_n}\Big) \\
[\![\text{unit}]\!]_X & = & (\mathsf{val} \mid X)\ \mathsf{unit} \\
[\![n]\!]_X & = & (\mathsf{val} \mid X)\ \mathsf{i}n \quad , \quad \forall n \in \mathcal{N}
\end{array}
$$

Figure 4.2: A translation of Ξ into Local Bigraphs.

$$\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(-_0)) \mid \mathsf{pairr}(\mathsf{exp}(-_1))) \\ \rightarrowtriangle \quad \mathsf{pair}(\mathsf{pairl}(\mathsf{val}(-_0)) \mid \mathsf{pairr}(-_1)) \tag{4.2}$$

$$\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(-_0)) \mid \mathsf{pairr}(\mathsf{val}(-_1))) \\ \rightarrowtriangle \quad \mathsf{val}(\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(-_0)) \mid \mathsf{pairr}(\mathsf{val}(-_1)))) \tag{4.3}$$

$$\mathsf{fst}(\mathsf{val}(\mathsf{pair}(\mathsf{pairl}(-_0) \mid \mathsf{pairr}(-_1)))) \quad \rightarrowtriangle \quad -_0 \tag{4.4}$$

$$\mathsf{snd}(\mathsf{val}(\mathsf{pair}(\mathsf{pairl}(-_0) \mid \mathsf{pairr}(-_1)))) \quad \rightarrowtriangle \quad -_1 \tag{4.5}$$

$$\mathsf{let}(\mathsf{letd}(\mathsf{val}(-_0)) \mid \mathsf{letb}_{(x)}(\mathsf{sub}_{(x)}(-_1\langle x\rangle)\langle x\rangle \mid \mathsf{def}_x(\mathsf{val}(-_0))) \tag{4.6}$$

$$\mathsf{ref}(\mathsf{val}(-_0)) \parallel \mathsf{store}(-_1) \\ \rightarrowtriangle \quad /l.\mathsf{val}(\mathsf{cell}_l) \parallel \mathsf{store}(\mathsf{cell}'_l(\mathsf{val}(-_0)) \mid -_1) \tag{4.7}$$

$$\mathsf{deref}(\mathsf{val}(\mathsf{cell}_l)) \parallel \mathsf{store}(\mathsf{cell}'_l(-_0) \mid -_1) \\ \rightarrowtriangle \quad -_0 \mid l/ \parallel \mathsf{store}(\mathsf{cell}'_l(-_0) \mid -_1) \tag{4.8}$$

$$\mathsf{assign}(\mathsf{acell}(\mathsf{val}(\mathsf{cell}_l)) \mid \mathsf{aval}(\mathsf{exp}(-_0))) \\ \rightarrowtriangle \quad \mathsf{assign}(\mathsf{acell}(\mathsf{val}(\mathsf{cell}_l)) \mid \mathsf{aval}(-_0)) \tag{4.9}$$

$$\mathsf{assign}(\mathsf{acell}(\mathsf{val}(\mathsf{cell}_l)) \mid \mathsf{aval}(\mathsf{val}(-_0))) \parallel \mathsf{store}(\mathsf{cell}'_l(-_1) \mid -_2) \\ \rightarrowtriangle \quad \mathsf{val}(\mathsf{unit}) \oplus \{l\} \parallel \mathsf{store}(\mathsf{cell}'_l(-_0) \mid -_2) \tag{4.10}$$

$$\mathsf{app}(\mathsf{appl}(\mathsf{val}(-_0)) \mid \mathsf{appr}(\mathsf{exp}(-_1))) \\ \rightarrowtriangle \quad \mathsf{app}(\mathsf{appl}(\mathsf{val}(-_0)) \mid \mathsf{appr}(-_1)) \tag{4.11}$$

$$\mathsf{app}(\mathsf{appl}(\mathsf{val}(\mathsf{lam}_{(x)}(-_0\langle x\rangle))) \mid \mathsf{appr}(\mathsf{val}(-_1))) \\ \rightarrowtriangle \quad \mathsf{sub}_{(x)}(-_0\langle x\rangle \mid \mathsf{def}_x(\mathsf{val}(-_1))) \tag{4.12}$$

$$\mathsf{app}(\mathsf{appl}(\mathsf{val}(\mathsf{fix}_{(f,x)}(-_0\langle f, x\rangle))) \mid \mathsf{appr}(\mathsf{val}(-_1))) \\ \rightarrowtriangle \quad \mathsf{sub}_{(f)}(\mathsf{sub}_{(x)}(-_0\langle x\rangle \mid \mathsf{def}_x(\mathsf{val}(-_1))) \mid \\ \qquad \mathsf{def}_f(\mathsf{val}(\mathsf{fix}_{(f,x)}(-_0\langle f, x\rangle)))) \tag{4.13}$$

$$\mathsf{C}(\mathsf{val}(-_0)) \quad \rightarrowtriangle \quad \mathsf{val}(\mathsf{C}(\mathsf{val}(-_0))) \tag{4.14}$$

$$\mathsf{case}(\mathsf{casel}(\mathsf{val}(\mathsf{C}(-_0))) \mid \mathsf{casee}_{(x)}(\mathsf{C}(-_1\langle x\rangle)) \mid -_2) \\ \rightarrowtriangle \quad \mathsf{sub}_{(x)}(-_1\langle x\rangle \mid \mathsf{def}_x(-_0)) \tag{4.15}$$

$$\mathsf{var}_x \parallel \mathsf{def}_x(\mathsf{val}(-_0)) \quad \rightarrowtriangle \quad \mathsf{val}(-_0) \mid \{x\} \parallel \mathsf{def}_x(\mathsf{val}(-_0)) \tag{4.16}$$

$$\mathsf{sub}_{(x)}(-_0 \mid \mathsf{def}_x(-_1)) \quad \rightarrowtriangle \quad -_0 \tag{4.17}$$

$$\mathsf{sub}_{(x)}(\mathsf{val}(\mathsf{lam}_{(y)}(-_0\langle x, y\rangle)) \mid \mathsf{def}_x(\mathsf{val}(-_1))) \\ \rightarrowtriangle \quad \mathsf{val}(\mathsf{lam}_{(y)}(\mathsf{sub}_{(x)}(-_0\langle x, y\rangle \mid \mathsf{def}_x(\mathsf{val}(-_1))))) \tag{4.18}$$

$$\mathsf{sub}_{(x)}(\mathsf{val}(\mathsf{fix}_{(f,y)}(-_0\langle x, f, y\rangle)) \mid \mathsf{def}_x(\mathsf{val}(-_1))) \\ \rightarrowtriangle \quad \mathsf{val}(\mathsf{fix}_{(f,y)}(\mathsf{sub}_{(x)}(-_0\langle x, f, y\rangle \mid \mathsf{def}_x(\mathsf{val}(-_1))))) \tag{4.19}$$

Figure 4.3: Reaction rules for Ξ.

immediately upon creation to only grant the freshly generated reference cell cell_l access to that store cell. Thus, we know that l is not in $cod(-_0), cod(-_1)$.

Dereferencing and assignment also work as one would expect, (4.8), (4.9) and (4.10), the extra outer name l being necessary to maintain the outer face. This prime product is always defined because we know that $-_0$ is a value and thus has outer width 1 (like $l/$). Location l is closed, but this is *not* explicit in rules (4.8)-(4.10) because the context has to be able to identify the name l of different regions. In the second assignment rule (4.10) we extend $\mathsf{val}(\mathsf{unit})$ with the wiring $\{l\} : \epsilon \to \{l\}$ to maintain l in the *outer* face of the first region of the reactum.

Applying the fixed-point construction to a value results in a nested substitution, as one would expect. We have a rule pair (4.14) and (4.15) for each declared constructor in the source program.

Rules (4.12), (4.16), and (4.17) are reminiscent of Milner's encoding of a λ-calculus with explicit substitutions into Local Bigraphs [Mil07]. Rule (4.12) is application of a lambda abstraction (to a value), rule (4.16) performs a substitution, and rule (4.17) discards an explicit substitution. The difference between our rules and those of [Mil07] is that we have val controls, and we have written the implicit prime product with set $\{x\}$ in rule (4.16).

In rule (4.16) we exploit Local Bigraphs by using a wide reaction rule in connection with binding to substitute "at a distance" one occurrence at a time. (This does not work in *Binding* Bigraphs because here a name can not reside in multiple locations. However, one can tediously obtain the same behaviour by means of more reaction rules.)

Rule (4.17) terminates the substitution when used; it can not be applied before we are actually done substituting, because $-_0$ can not contain x in its outer face since in that case $cod(R) \neq cod(R')$.

Rules (4.18) and (4.19) propagate explicit substitutions under lambda abstractions and fixed-point constructs. We will discuss this further in Section 4.4.

Notice how we have written two sites as $-_0$ in R' implicitly defining $\vec{\iota} = \{0, 1 \mapsto 0\}$. The rest of the rules should be clear.

4.3.1 Encoding of natural numbers

Here we present an encoding of natural numbers in Bigraphs using a Peano-like representation with zero and successor. We also encode rules implementing simple operations like equality on these using an encoding of Boolean constants True and False. There should be no surprises in Figure 4.4. The rules presented in Figure 4.5 should be straightforward, but do notice

Control	Activity	Arity	Comment
true	atomic	$0 \to 0$	True
false	atomic	$0 \to 0$	False
z	atomic	$0 \to 0$	Zero (i0)
s	active	$0 \to 0$	Successor
argl	active	$0 \to 0$	Left argument
argr	active	$0 \to 0$	Right argument
eqi	active	$0 \to 0$	Equality operator
lti	active	$0 \to 0$	Less than operator
addi	active	$0 \to 0$	Addition operator
subi	active	$0 \to 0$	Subtraction operator
muli	active	$0 \to 0$	Multiplication operator

Figure 4.4: Signature for natural numbers encoding in Local Bigraphs.

that we use non-negative subtraction on integers (so really we restrict ourselves to natural numbers) in rule (4.31) and that multiplication is computed according to the equation $(m+1)*(n+1) = (n+1)+m*(n+1)$ in rule (4.36). As mentioned in Chapter 4 we introduce a shorthand notation for natural numbers; i0 means z, i1 means s z, i2 means s (s z), and so forth.

In an implementation of a bigraphical rewrite engine, it is probably too inefficient to work with bigraphical integers (and Booleans) so one would perhaps choose to work with the integers (and Booleans) of the implementation language instead. Using i−controls is a more human-readable representation.

4.3.2 An example exploring references

In this chapter we wish to explore the encoding of references, and this only uses a fragment of Ξ. We shall return to the other parts of Ξ in Chapter 5. Consider the program shown in Figure 4.6. It is written in Ξ, and translated into Bigraphs. Clearly, the result of evaluating this program should be the value 4 (i4). This program uses *aliasing*, and we intend to see if the encoding behaves correctly. We have presented the bigraph in short-hand notation leaving wirings implicit.

By A_0 we denote the bigraph shown in Figure 4.6. The initial state A_0 rewrites to $A_{14} = /l.\mathsf{val}(\mathsf{i4}) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i4})))$ via the reduction path (4.7) − (4.6) − (4.16) − (4.6) − (4.16) − (4.17) − (4.9) − (4.10) − (4.2) − (4.16) − (4.17) − (4.8) − (4.3) − (4.5). The most interesting steps are the uses of rules (4.7), (4.10), and (4.8), but we also show how rules (4.6), (4.16), and (4.17) are

$$\mathsf{eqi}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(\mathsf{z})) \rightarrowtriangle \mathsf{true} \tag{4.20}$$

$$\mathsf{eqi}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{s}(-_1))) \rightarrowtriangle \mathsf{eqi}(\mathsf{argl}(-_0) \mid \mathsf{argr}(-_1)) \tag{4.21}$$

$$\mathsf{eqi}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(\mathsf{s}(-_0))) \rightarrowtriangle \mathsf{false} \tag{4.22}$$

$$\mathsf{eqi}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{z})) \rightarrowtriangle \mathsf{false} \tag{4.23}$$

$$\mathsf{lti}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(\mathsf{z})) \rightarrowtriangle \mathsf{false} \tag{4.24}$$

$$\mathsf{lti}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{s}(-_1))) \rightarrowtriangle \mathsf{lti}(\mathsf{argl}(-_0) \mid \mathsf{argr}(-_1)) \tag{4.25}$$

$$\mathsf{lti}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(\mathsf{s}(-_0))) \rightarrowtriangle \mathsf{true} \tag{4.26}$$

$$\mathsf{lti}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{z})) \rightarrowtriangle \mathsf{false} \tag{4.27}$$

$$\mathsf{addi}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(-_0)) \rightarrowtriangle -_0 \tag{4.28}$$

$$\mathsf{addi}(\mathsf{argl}(-_0) \mid \mathsf{argr}(\mathsf{z})) \rightarrowtriangle -_0 \tag{4.29}$$

$$\begin{aligned} &\mathsf{addi}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{s}(-_1))) \\ &\quad \rightarrowtriangle \quad \mathsf{s}(\mathsf{addi}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(-_1))) \end{aligned} \tag{4.30}$$

$$\mathsf{subi}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(-_0)) \rightarrowtriangle \mathsf{z} \tag{4.31}$$

$$\mathsf{subi}(\mathsf{argl}(-_0) \mid \mathsf{argr}(\mathsf{z})) \rightarrowtriangle -_0 \tag{4.32}$$

$$\mathsf{subi}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{s}(-_1))) \rightarrowtriangle \mathsf{subi}(\mathsf{argl}(-_0) \mid \mathsf{argr}(-_1)) \tag{4.33}$$

$$\mathsf{muli}(\mathsf{argl}(\mathsf{z}) \mid \mathsf{argr}(-_0)) \rightarrowtriangle \mathsf{z} \tag{4.34}$$

$$\mathsf{muli}(\mathsf{argl}(-_0) \mid \mathsf{argr}(\mathsf{z})) \rightarrowtriangle \mathsf{z} \tag{4.35}$$

$$\begin{aligned} &\mathsf{muli}(\mathsf{argl}(\mathsf{s}(-_0)) \mid \mathsf{argr}(\mathsf{s}(-_1))) \\ &\quad \rightarrowtriangle \quad \mathsf{addi}(\mathsf{argl}(\mathsf{s}(-_1)) \mid \mathsf{argr}(\mathsf{muli}(\mathsf{argl}(-_0) \mid \mathsf{argr}(\mathsf{s}(-_1))))) \end{aligned} \tag{4.36}$$

Figure 4.5: Operations on integers in Local Bigraphs.

matched because there are some subtleties here. We only show the maps $\eta, \vec{\iota}$ in the matches where they are important and non-trivial. The reader may want to look up these rules when inspecting the matches in what follows.

Notation 4.10. *We use S, T, U to denote certain sub-terms of a bigraph when these sub-terms are not important for the particular match, to increase readability when presenting the matches.*

```
let z = ref 5
in let y = z in
snd (y:=4,!z)

------------------------------------------------

let(letd(ref(val(i5))) |
    letb_(z)(
      let(letd(var_z) |
          letb_(y)(
            snd(pair(
                  pairl(assign(
                          aloc(var_y) |
                          aval(exp(i4)))) |
                  pairr(exp(deref(var_z))))))))) |
store()
```

Figure 4.6: A Ξ-program and its translation into Bigraphs.

We begin to evaluate the declaration part of the outermost let; A_0 matches (4.7) because we find

$$\begin{aligned} a &= \mathsf{i5} \parallel 1 : (\emptyset,\emptyset) \\ \omega &= \mathsf{id}_2 : (\emptyset,\emptyset) \to (\emptyset,\emptyset) \\ C &= \mathsf{let}\Big(\mathsf{letd}(-_2) \mid \mathsf{letb}_{(z)}(S)\Big) \mid -_3 : (\emptyset,\emptyset) \to \emptyset \\ a &= a' \\ \omega' &= 1 \otimes (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) : (\emptyset,\emptyset) \to (\emptyset,\emptyset) \end{aligned}$$

where S denotes the content of the outermost letb in A_0. It is easy to convince oneself that $A_0 = C \circ (R \oplus \omega) \circ a$. Consider an A_1 such that $A_0 \twoheadrightarrow A_1$:

$$A_1 = /l.\mathsf{let}\Big(\mathsf{letd}(\mathsf{val}(\mathsf{loc}_l)) \mid \mathsf{letb}_{(z)}(S)\Big) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5})))$$

Notice how in A_1 the closure is done "outermost" whereas in the rule it is done in R'. This situation is essentially the question whether

$$\Big(\mathsf{F}(-_0) \mid \mathsf{G}(-_1)\Big) \circ /l.\mathsf{H}_l \mid \mathsf{I}_l = /l.\mathsf{F}(\mathsf{H}_l) \mid \mathsf{G}(\mathsf{I}_l)$$

for some ions $\mathsf{F}, \mathsf{G}, \mathsf{H}, \mathsf{I}$. Looking at how composition is defined in Definition 8.3 (precategory of link graphs) of [JM04], we see that case one of the

definition of the composite link map applies so indeed the equation holds, and this also applies to our setting. (It also holds for ‖.) Thus, $A_1 = C \circ (R' \oplus \omega') \circ a'$ as required. Intuitively, it may seem weird that one can plug in two link-connected regions (when ‖ is used) into two separate holes and keep the connection intact – especially when one looks at the graphical representation. However, the closed link is a visual deception – formally the link map just maps the outer name l of H and I to an edge, and this relationship is maintained during composition.

We continue by showing how A_1 matches (4.6). We have $\eta = \mathsf{Id}_2$ and $\vec{\iota} = (\mathsf{Id}_\emptyset, \mathsf{Id}_x)$. When matching we can, without loss of generality, rename z to x in A_1 because the two terms denote the same bigraph.

$$\begin{aligned} a &= \mathsf{loc}_l \parallel S : (l, x) \\ \omega &= \mathsf{id}_l \mid \mathsf{id}_\emptyset : (l, \emptyset) \to l \\ C &= /l.\mathsf{id}_l \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5}))) : l \to \emptyset \\ a' &= a \\ \omega' &= \omega \end{aligned}$$

$A_2 = /l.\mathsf{sub}_{(z)}\Big(\mathsf{let}(\mathsf{letd}(\mathsf{var}_z) \mid \mathsf{letb}_{(y)}(S)) \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5})))$.

Next up is application of the substitution rule (4.16). We have $\eta = \{0, 1 \mapsto 0\}$ and $\vec{\iota} = (\mathsf{Id}_\emptyset, \mathsf{Id}_\emptyset)$.

$$\begin{aligned} a &= \mathsf{loc}_l : l \\ \omega &= l/ \parallel \mathsf{id}_l : l \to (l, l) \\ C &= /l.\mathsf{sub}_{(x)}\Big(\mathsf{let}(\mathsf{letd}(-_2\langle x, l\rangle) \mid \mathsf{letb}_{(y)}(S)) \mid -_3\langle x, l\rangle\Big) \\ &\qquad \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5}))) : (\{x, l\}, \{x, l\}) \to \emptyset \\ a' &= \mathsf{loc}_l \parallel \mathsf{loc}_l : (l, l) \\ \omega' &= \mathsf{id}_l \parallel \mathsf{id}_l : (l, l) \to (l, l) \end{aligned}$$

So, A_3 is:

$$\begin{aligned} A_3 = {} & /l.\mathsf{sub}_{(z)}\Big(\mathsf{let}(\mathsf{letd}(\mathsf{val}(\mathsf{loc}_l)) \mid \mathsf{letb}_{(y)}(S)) \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big) \\ & \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5}))) \,. \end{aligned}$$

Now, we rewrite A_3 with rule (4.6) and then rule (4.16) to obtain

```
A_5 = /l . sub_(z)(
            sub_(y)(snd(pair(
```

```
                    pairl(
                      assign(
                        aloc(val(loc_l)) |
                        aval(exp(val(i4))))) |
                    pairr(exp(deref(var_z))))) |
              def_y(val(loc_l))) |
        def_z(val(loc_l))) |
  store(loc'_l(val(i5))) .
```

These two steps unfolded the innermost let to a sub, and then substituted in the value of x for y. We see that there are no more free occurrences of the name y so we can terminate that substitution with rule (4.17). Notice that $\eta = \mathsf{Id}_0$.

$$
\begin{aligned}
a &= T \parallel \mathsf{val}(\mathsf{loc}_l) : (\{z,l\},l)\\
\omega &= \mathsf{id}_{\{z,l\}} \mid \mathsf{id}_l : (\{z,l\},l) \to \{z,l\}\\
C &= /l.\mathsf{sub}_{(z)}\Big(-_2\langle z,l\rangle \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5}))) : \{z,l\} \to \emptyset\\
a' &= T : \{z,l\}\\
\omega' &= \mathsf{id}_{\{z,l\}} : \{z,l\} \to \{\, z,l\}
\end{aligned}
$$

where T denotes the sub-term $\mathsf{snd}(\ldots)$ of A_5. State A_6 is:

$$A_6 = /l.\mathsf{sub}_{(z)}\Big(T \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i5}))) \,.$$

Now, simply remove the exp from aval in T with (4.9) to obtain:

```
A_7 = /l . sub_(z)(snd(pair(
                      pairl(assign(
                              aloc(val(loc_l)) |
                              aval(val(i4)))) |
                      pairr(exp(deref(var_z))))) |
                def_z(val(loc_l))) |
          store(loc'_l(val(i5))) .
```

A_7 can be rewritten with (4.10) as follows. $\eta = \{0 \mapsto 0, 2 \mapsto 2\}$.

$$
\begin{aligned}
a &= \mathsf{i4} \parallel \mathsf{val}(\mathsf{i5}) \parallel 1 : (\emptyset,\emptyset,\emptyset)\\
\omega &= \mathsf{id}_\emptyset \parallel (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) : (\emptyset,\emptyset,\emptyset) \to (\emptyset,\emptyset)\\
C &= /l.\mathsf{sub}_{(z)}\Big(\mathsf{snd}\Big(\mathsf{pair}(\mathsf{pairl}(-_3\langle l\rangle) \mid \mathsf{pairr}(U))\Big) \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big) \mid -_4\langle l\rangle\\
&\quad : (l,l) \to \emptyset\\
a' &= \mathsf{val}(\mathsf{i4}) \parallel 1 : (\emptyset,\emptyset)\\
\omega' &= 1 \parallel (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) : (\emptyset,\emptyset) \to (\emptyset,\emptyset)
\end{aligned}
$$

The result is state A_8:

$$A_8 = /l.\mathsf{sub}_{(z)}\Big(\mathsf{snd}\Big(\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(\mathsf{unit})) \mid \mathsf{pairr}(U))\Big) \mid \mathsf{def}_z(\mathsf{val}(\mathsf{loc}_l))\Big)$$
$$. \qquad \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i4})))$$

Notice how the store has changed. It is here that we realise why the link l has to be open in the definition of (4.10). Had it been closed we would not be able to supply a proper context C because there would be no way to identify the location inside def_z with the locations inside R (that are plugged into the holes $-_3, -_4$), since these would be in an already closed link, and thus not accessible (by composition) from the outside. This is an important property of closed links. We may now rewrite A_8 to

$$A_{11} = /l.\mathsf{snd}\Big(\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(\mathsf{unit})) \mid \mathsf{pairr}(\mathsf{deref}(\mathsf{val}(\mathsf{loc}_l))))\Big)$$
$$\mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i4})))$$

by the non-controversial sequence (4.2) - (4.16) - (4.17). Finally, we may rewrite using (4.8).

$$\begin{aligned}
a &= \mathsf{val}(\mathsf{i4}) \parallel 1 : (\emptyset, \emptyset) \\
\omega &= 1 \parallel (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) : (\emptyset, \emptyset) \to (\emptyset, \emptyset) \\
C &= /l.\mathsf{snd}(\mathsf{pair}(\mathsf{pairl}(\mathsf{val}(\mathsf{unit})) \mid \mathsf{pairr}(-_3\langle l\rangle))) \mid -_4\langle l\rangle : (l, l) \to \emptyset \\
a' &= \mathsf{val}(\mathsf{i4}) \parallel \mathsf{val}(\mathsf{i4}) \parallel 1 : (\emptyset, \emptyset, \emptyset) \\
\omega' &= \mathsf{id}_\emptyset \parallel (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) : (\emptyset, \emptyset, \emptyset) \to (\emptyset, \emptyset)
\end{aligned}$$

Again, the rule would not have worked with a closed link had there been a dereference operation inside pairl of A_{11}, for example. In this case, however, it works out either way. We finish evaluation by trivially rewriting with (4.3) and then (4.5). Thus, we have $A_{14} = /l.\mathsf{val}(\mathsf{i4}) \mid \mathsf{store}(\mathsf{loc}'_l(\mathsf{val}(\mathsf{i4})))$ as promised.

4.4 Dynamic correspondence

We have argued by example that there is a dynamic correspondence between Ξ and the encoding hereof in Local Bigraphs. Informally, we claim that every time we reduce a term in Ξ by $\longrightarrow$ we can mimic that in the bigraphical reactive system with one or more uses of $\multimap$, because substitutions are explicit and we need to propagate these (and also lift exp controls).

Consider the following example, where we have omitted controls val, appl, and appr, to aid readability:

$$\begin{aligned} & \mathsf{app}(\mathsf{app}(\mathsf{lam}_{(x)}(\mathsf{lam}_{(y)}(\mathsf{var}_x)) \mid \mathsf{exp}(\mathsf{i4})) \mid \mathsf{exp}(\mathsf{i2})) \\ \rightarrow\!\!\triangleright^2 \quad & \mathsf{app}(\mathsf{sub}_{(x)}(\mathsf{lam}_{(y)}(\mathsf{var}_x) \mid \mathsf{def}_x(\mathsf{i4})) \mid \mathsf{exp}(\mathsf{i2})) \\ \rightarrow\!\!\triangleright \quad & \mathsf{app}(\mathsf{lam}_{(y)}(\mathsf{sub}_{(x)}(\mathsf{var}_x \mid \mathsf{def}_x(\mathsf{i4}))) \mid \mathsf{exp}(\mathsf{i2})) \\ \rightarrow\!\!\triangleright^2 \quad & \mathsf{sub}_{(y)}((\mathsf{sub}_{(x)}(\mathsf{var}_x \mid \mathsf{def}_x(\mathsf{i4}))) \mid \mathsf{def}_y(\mathsf{i2})) \end{aligned}$$

And now there are three reaction (computation) paths leading to i4 depending on the order of performed and discarded substitutions. In this case the order does not matter as there is confluence. This example illustrates how the rules (4.18) and (4.19) propagate substitutions so that we may have the desired dynamic correspondence.

To formally prove the dynamic correspondence we would need a substitution lemma, a lemma allowing us to separate a context from a redex, and a lemma corresponding to Definition 4.8, but for Bigraphs. We comment a little further on the desired theorem in Section 8.1.6. Proving that the correspondence holds both ways would likely require us to keep track of which bigraphs are actually images of Ξ-terms.

We conclude that references can be encoded in Local Bigraphs using closed links, that they seem to behave as desired, and conjecture that this can be proved formally.

4.5 Discussion

One may wonder, intuitively, why we chose to encode references via closed links and not binders. We give such an intuition here.

Closed links were included in the Bigraphs to capture the notion of *name restriction* from the π-calculus. Binding ports likewise capture the π-calculus notion of *prefix*. In essence, the difference between closed links and binding ports in (Binding and Local) Bigraphs is *locality* and the *scope rule*. All binders are located (while edges are not) and have to obey the scope rule, i.e., all peers of a certain binder must be located beneath that binder in the place graph, roughly said.[2] This means that using binders is in general more restrictive, which may be preferred in some situations like when modelling *access control*. In our case we have a *global* store located at top level. Thus, in this case, it should be possible to encode references via a

[2] We merely aim to provide some intuition and thus refer the reader to [JM04] for precise definitions.

top-level vertex acting like a store by having a binding port for each created location.

A reason for using binders could be to investigate whether there is a simpler proof of the dynamic correspondence between Ξ and the encoding, than with free closed links. The idea is to define a family of controls store^m, indexed by an ordinal m defining the set of (binding) ports on store. The program resides within this control. Initially, we have $\mathsf{store}^0(\ldots)$ with no ports meaning that no locations have been created. Then, every time a location is created we replace the current store^i with a new one, namely store^{i+1}, and bind this new location to the new port (keeping the old bindings). When binding a new location to the store we also create a value node holding the value of the newly created location, and link this to the new binding port of the store. This idea resembles one of Robert Harper [Har00] (Chapter 16), though in another setting.

Summing up, we can say that closed links were chosen because they do not have locality, which enables us to avoid worrying about the locality of the store holding the references, which is the natural way to think about a store.

5

A Real-life Location Model

In this chapter we make more precise what is meant by the term "reflective" building. Having a clear idea of which properties such a building should possess we move on to exhibit a Plato-graphical model corresponding to such a building. Some parts of the model are formulated directly in Bigraphs, and some parts are expressed in a slightly enhanced and sugared form of Ξ, named Ξ_{sugar}. In a sense, this chapter brings together the work of the Chapters 2, 3, and 4.

5.1 A reflective building

We define a reflective building in more detail. In [Hop00] on "sentient computing" the following three "location categories" are stated as important.

- Containment
- Proximity
- Coordinate systems

Containment refers to an object being within a container (location). Proximity is the notion of being close to something. Coordinate systems provide location in space (subject to some error value). We have seen containment before as a spatial relationship on locations. We interpret proximity to cover two situations: (1) Near measured in physical distance, and (2) near as within the same symbolic range, that is, within the same container (at some level in the topology/hierarchy). Our starting point is a symbolic model so we abstract away from coordinates in this treatment.

We do, however, add some additional properties to our list of reflective building properties:

- Device positioning
- A fixed set of uniquely identifiable mobile devices

When considering a reflective building we think of located-objects as being mobile devices (and henceforth refer to them as such) such a mobile phones and PDAs.

In our modelling effort we abstract the reflective building by abstracting over location types (floors, wings, rooms etc.) and device types (PDA, mobile phone, laptop etc.).

We capture systems where, e.g., a user arrives at a museum and receives a mobile device to be used together with the positioning system and the location model at this museum.

5.2 The model

Before the presentation of the Plato-graphical model it is instructive to explain the key design choices made.

5.2.1 Design choices

Several choices have been made regarding the languages used, the representation of locations and devices, and the location hierarchy. We treat these considerations in turn.

Languages used

We have used two languages in the model. The parts **C** and **S** of the model are written using the bigraphical term language, and parts **L** and **A** in a sugared version Ξ_{sugar} of the calculus Ξ presented in Chapter 4. We prefer to work within Bigraphs except when it becomes too inconvenient. We show that Bigraphs are well-suited for modelling **C** and **S**, and not just **C** as suggested in Chapter 3. The location model **L** is, however, sufficiently complicated to write natively in Bigraphs for us to prefer writing it in Ξ_{sugar}. **A** communicates with **L** so it makes sense to write **A** in Ξ_{sugar}, apart from the fact that it is easier to program **A** in Ξ_{sugar} than natively in Bigraphs, as will become apparent. Thus, we have a real world **C** written in Bigraphs, a proxy $\mathbf{P} = \mathbf{S} \parallel \mathbf{L}$ spanning both Bigraphs and Ξ_{sugar}, and a location-aware application **A** written in Ξ_{sugar}.

A key issue in this setup is how to realise the communication between the "bigraph world" and the "Ξ_{sugar} world" in **P**. Later in this chapter we explain how Ξ_{sugar}-programs correspond to Ξ-programs.

Representing locations and devices in C (and S)

We use one control $\mathsf{loc} : 0 \to 0$ to represent locations and one $\mathsf{dev} : 0 \to 0$ to represent devices abstracting away different location types. Another option is to use different controls for the "different" types of locations in a building; wing, floor, room and so forth. The reasons for choosing one loc control are (1) to limit the number of reaction rules to be written – we do not need to have a rule for each combination of the different location (and device) types where one location is source and the other target, and (2) identity and type of a location can be represented conveniently in another way (to which we return below). Devices are simply represented by a device control.

Known devices

As mentioned in Section 5.1 we consider a reflective building to have a fixed set of known devices, which pertains to all parts of the model. Some may be in use and some may not. A device is in use when it is in a location which is not the special "unused devices" location. Initially, each device is either in this location or in one of the locations of the location hierarchy (to be addressed next). This property should be invariant under reaction. No new devices can appear while the system is running. As mentioned in Section 5.1 this is a realistic choice, and technically it significantly simplifies our task. This is due to the fact that when a device is discovered in **C** we have to mirror this by a discovery in **L** via **S**, and thus generate a corresponding fresh identifier for this device in **L**. It is difficult to ensure this automatically.

A static tree-structured location hierarchy

It is a simplifying choice to work with a static location hierarchy as opposed to a dynamic one. It is reasonable to have a static location hierarchy in this case because a (reflective) building in the real world seldom changes. Thus, the location hierarchy in the model needs to be altered only on rare occasions. According to [Leo98] location hierarchies are typically static. Furthermore, we organise locations in a tree utilising the structure of place graphs. This is a limiting choice, but not an uncommon one. The same hierarchy is present in **C** and **L**, but represented differently.

Modelling the real world

For our purposes the following abstraction is suitable: Devices can enter, move around in, and leave a (reflective) building which is observationally equivalent to being turned off. These reconfigurations should be modelled in **C**.

Identification of locations and devices in C and L

A location in the bigraphical part of the model can be identified by a link or by an embedded identity control. We assume that locations are uniquely identified which requires an equality operation on their identifiers. Such identifiers could, e.g., be natural numbers or strings. We have chosen natural numbers for simplicity and to keep our focus. Strings can also be encoded. The property that no two locations or devices have the same identifier is invariant under reaction, i.e., maintained by every reaction rule.

Now, consider the two approaches; identification via a link or an embedded control. Links can be open (outer name) or closed (edge). Using closed links in **C** is not a viable solution because they do not in fact reveal any identity information to the context and thus can not be distinguished, which is required to support certain basic queries. Open links could be used because the location hierarchy is static; location identifiers are assumed to be unique initially, and new locations are not introduced under reaction. Thus, there is no risk of an identifier (outer name) being reused, i.e., two different locations being identified. As no new devices are introduced we are safe. The same reasoning applies to device identifiers.

Another option is to place a unique identity control as a child of each location and device control, which is what we choose. Our choice is supported by the following two considerations: (1) The model becomes simpler. Had we chosen link identifiers in **C** we would have to relate these links to location identifiers in **L** which could be natural numbers (or strings). This can be done by "exporting" the location identifiers of **L** to top level of the Plato-graphical system, which is not entirely straightforward as will become apparent later. (2) Given the intuition that controls represent entities and links represent (wireless) connections between entities it seems appropriate to model identity of an entity as part of that entity (e.g. like a MAC address of a device or the name of a room), i.e., via an identity control. Initially, **C** and **L** have the same location hierarchy (in their respective languages). Next we discuss how closely coupled **C** and **L** should be.

Relating C and L via S

An argument for a high degree of coupling between **C** and **L** is a low modelling effort, whereas a low degree of coupling implies stronger modularity, which is an argument against a tight coupling. Let us consider the setup to decide on a design choice here. The setup is that the initial configuration of the system is given, and in particular that the location hierarchies in **C** and **L** are coherent. For simplicity, take locations and devices to be identified by Ξ-integers in **L**. Encoding integers (or rather natural numbers) in Bigraphs yields an easy correspondence, defined in **S**, between identifiers in **C** and **L**. This is a rather tight coupling, but not an unreasonable one since the user of the system is obliged to define the location hierarchy and the devices in both **C** and **L** initially. Furthermore, decoupling **C** and **L** merely comes down to a mapping in **S**.

Location systems as Plato-graphical systems

Consider Figure 5.1. Location-aware applications (agents) are captured by

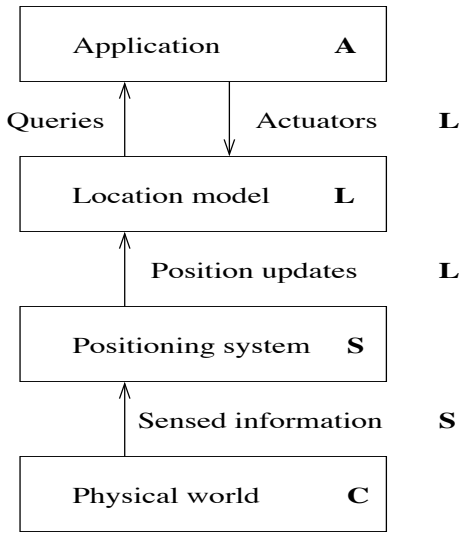


Figure 5.1: Overall location system model seen as Plato-graphical.

the part **A**. The location model including queries (and actuators) is part **L**. We have included both queries and actuators in **L** because they are really interfaces to the location model. The positioning system including sensors

is part **S**. **S** informs **L** of location updates. The physical world is part **C**.

5.2.2 *Introducing the model*

We begin by briefly recalling Plato-graphical models. Consider a Plato-graphical model $\mathcal{X} = (\mathbf{C_X}, \mathbf{P_X}, \mathbf{A_X})$ as defined in Chapter 3. This model consists of a world $\mathbf{c}_X \in \mathbf{C_X}$, a proxy $\mathbf{P_X}$ with constituents $\mathbf{s}_X \in \mathbf{S_X}$ (sensor) and $\mathbf{l}_X \in \mathbf{L_X}$ (the location-aware system's representation of the world), and a location-based application $\mathbf{a}_X \in \mathbf{A_X}$. The notation used, e.g. $\mathbf{c}_X \in \mathbf{C_X}$, signifies that $\mathbf{c}_X$ is the state of the BRS $\mathbf{C_X}$.

The overall intuition about the system is as follows: $\mathbf{c}_X$ reconfigures as it pleases. Essentially, we have three reconfigurations in $\mathbf{C_X}$; discovery of a device in some location, movement of a device from a location to a parent or sub-location, and loss of a device (the system loses track of the device). Much like in the real world, changes in $\mathbf{c}_X$ occur in an unpredictable and non-deterministic manner. The system $\mathbf{S_X}$ *observing* the state $\mathbf{c}_X$ of the world tries as best it can to *inform* $\mathbf{l}_X$ of change to $\mathbf{c}_X$ by invoking certain "interface functions" in $\mathbf{L_X}$ to update its internal representation of the world (building). $\mathbf{S_X}$ has access to $\mathbf{C_X}$ and $\mathbf{L_X}$ by shared controls (with $\mathbf{C_X}$) and an outer name (with $\mathbf{L_X}$). Due to the asynchronicity between $\mathbf{C_X}$ and $\mathbf{L_X}$ we are likely to experience some discrepancy in the states of the two parts. This is perfectly realistic with respect to real-life indoor positioning systems such as Ekahau.

To make our model even more realistic we envision introducing time so that events from $\mathbf{S_X}$ to $\mathbf{L_X}$ can be timestamped and thus ordered. This should enable $\mathbf{L_X}$ to update its internal representation to more accurately match $\mathbf{c}_X$ as observed by $\mathbf{S_X}$. $\mathbf{A_X}$ can query $\mathbf{L_X}$ via a specific set of "interface functions". Recall that $\mathbf{C_X}$ and $\mathbf{S_X}$ are native BRSs whereas $\mathbf{L_X}$ and $\mathbf{A_X}$ are the BRSs resulting from a translation of Ξ_{sugar}-programs into Local Bigraphs along with the bigraphical reaction semantics defined in Chapter 4. Thus using the "multi-lingual feature" of Plato-graphical models.

Having briefly given the intuition behind the workings of the model we proceed to define it as a Plato-graphical model.

Our reflective building

We choose an example which holds the essential conceptual challenges arising from the previous discussion, but it is also kept simple for the sake of clarity. A more complex building, e.g. the IT University of Copenhagen, can likely be modelled without problems.

Introducing the simple reflective building We model a piece of ITU. Before showing the building as a tree and a bigraph, we briefly state the intention. We model the building, the atrium (in reality spanning all five floors) in the centre of the building, two of the four wings, two of the six floors, a hallway, and two rooms. The building (i1) contains the atrium (i2) and the wings B (i3) and C (i4). Wing C spans two floors, namely the third floor (i5) and the fourth floor (i6). The fourth floor contains a hallway (i7) which in turn contains two rooms, namely room 4C16 (i8) and 4C10 (i9). We assume six known devices numbered i10 through i15, all in use. One device is in the building, two devices are in the atrium, one is on the third floor, one is in room 4C16, and one is in room 4C10.

We could have made other choices, which we address shortly, but for now, we ask the reader to consider the informal graphical representation in Figure 5.2. Do notice that the id-controls around location identifiers have been left out for simplicity. Some choices were made to arrive at exactly the location hierarchy of Figure 5.2, each one is addressed in turn.

- The ordering of wings and floors.
- Where devices can reside.

Of course, we could have included more wings, floors, rooms etc. easily. First of all, notice how wings are considered to be above floors in the hierarchy. This indicates that one has to be within a wing to move from one floor to another. At the ITU we have stairways that allow this movement. However, it is also possible to move from a wing to the building, into an elevator, and to another floor. This is not possible in this model, but we consider it to be a cosmetic problem that is irrelevant with respect to our purposes at present. Switching the ordering of wings and floors would yield a similar problem. Thus, we can either stick with an ordering and accept the limitation, or model the location hierarchy using two trees (views), one with wings above floors and the other with floors above wings. That would, however, complicate things because devices would then have to be situated in both trees and make a movement in both trees at once. This movement would be between siblings in one tree, but not in the other tree. As we argue below, the movement between siblings is the most reasonable and realistic having chosen a tree structure to organise locations. Alternatively, links could be used to indicate paths between locations, but that would defeat the purpose of having imposed a structure on locations in the first place.

A brief detour: The link graph is usable for supporting the connected-to and distance relationships of Chapter 2 that allow for nearest neighbour

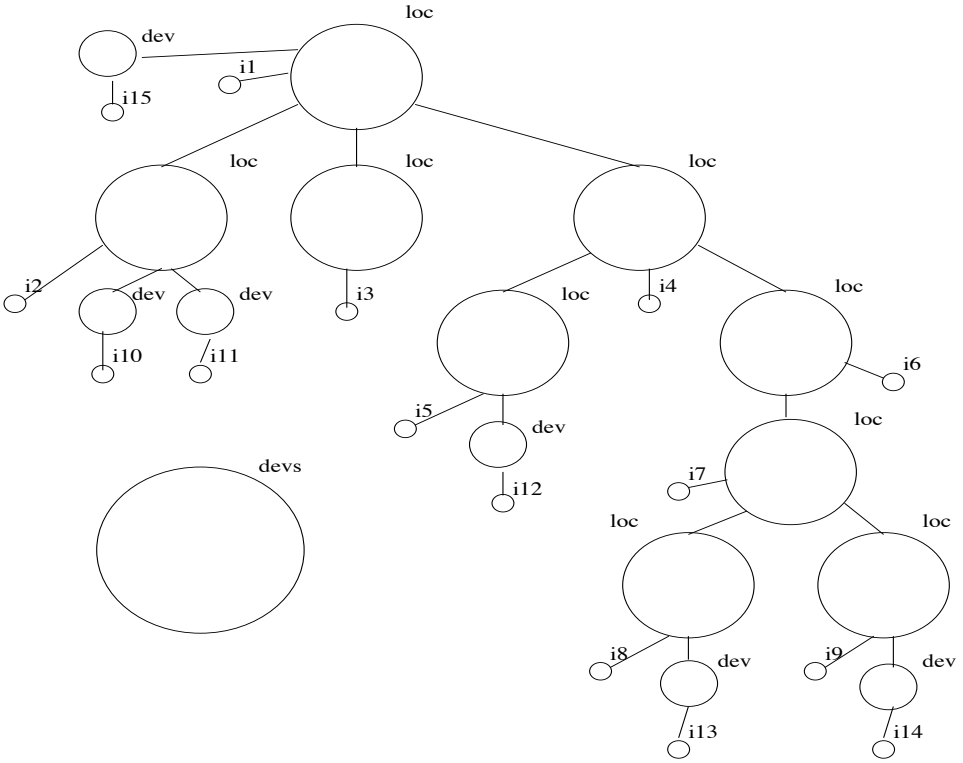


Figure 5.2: The building β as a tree where the id-controls around location identifiers have been left out, for simplicity.

and navigation queries.

We have decided that devices can reside in any location, and that locations can contain both devices and other locations. This is flexible and does not cause any modelling problems.

The reader can easily imagine the containment relation of Fig. 5.2. As a bigraph, identifiers (natural numbers) would be represented by nodes with atomic controls and depicted as black boxes.We proceed by presenting the model as a Plato-graphical system.

5.2.3 $\mathcal{X}$ as a Plato-graphical model

We divide the presentation into four parts: The native bigraphical parts (1) $\mathbf{C}_{\mathcal{X}}$ and (2) $\mathbf{S}_{\mathcal{X}}$, and the Ξ_{sugar}-parts (3) $\mathbf{L}_{\mathcal{X}}$ and (4) $\mathbf{A}_{\mathcal{X}}$.

Recall that we work in Local Bigraphs so the arity of a control is *binding* → *free*. Furthermore, we consider *activity* and *atomicity* to be integral parts of Bigraphs, but as shown in [Jen07] activity and atomicity could be considered *place-sortings* on *basic* bigraphs. We refer to [Jen07] for the details.

The world part $\mathbf{C_X}$

We begin by showing the building β as a bigraph in our term language (as before, omitting details such as identity wirings etc.), and then proceed to define the signature and dynamics of the BRS $\mathbf{C_X}$.

```
loc(i1 | dev(i15) |
      loc(i2 | dev(i10) | dev(i11)) |
      loc(i3) |
      loc(i4 | loc(i5 | dev(i12)) |
               loc(i6 | loc(i7 | loc(i8 | dev(i13)) |
                                 loc(i9 | dev(i14))))))
| devs()
```

As mentioned, β is a tree. Having made that choice it is reasonable to model device movement by allowing devices to move from a location l_2 into a sub-location l_3 of l_2, or into the parent location l_1 of l_2. To justify this claim, think of β; device i13 is situated in location 4C16 (i8), and the only doorway of 4C16 is into the hallway (i7), and the hallway does allow movement into another office, namely 4C10 (i9). All devices are in use in this start configuration. Remember that a device is either in a location or in devs.

The signature and dynamics of $\mathbf{C_X}$ are defined as follows in Figure 5.3; context $\mathbf{C_X} = (\mathcal{K}_{\mathbf{C_X}}, \mathcal{R}_{\mathbf{C_X}})$. First, notice how we have allowed ourselves to represent natural numbers by i−controls instead of using the more low-level representation with zero and successor. Equality and other basic operations can be implemented by a countably infinite set of reaction rules, capturing all combinations. Rule (5.1) discovers a device by moving it from devs to some loc. The rule is parametrised over the identity of the location, its content before the discovery, the set of known devices, and the identity of the device being discovered. Rule (5.2) performs the opposite operation, namely to lose track of a device. Notice how a device can be discovered and lost in any location which allows us to model a device being switched off (manually or because the battery runs out, e.g.) and turned on again in another location by rule (5.1). The rules (5.1) and (5.2) are dual in a sense. The rules (5.3) and (5.4) are likewise dual and move a device up or down one step in the location tree. We argued earlier that this is a fair representation

Context $\mathbf{C_X}$.

Control	Activity	Arity	Comment
id	passive	$0 \to 0$	Hosts identifier control
loc	passive	$0 \to 0$	Possibly nested location
dev	passive	$0 \to 0$	Mobile device
devs	passive	$0 \to 0$	Hosts unused devices
i0,i1,i2...	atomic	$0 \to 0$	Infinite family of identifiers

$$\mathsf{loc}(-_0) \parallel \mathsf{devs}(-_1 \mid \mathsf{dev}(-_2)) \rightarrow \mathsf{loc}(-_0 \mid \mathsf{dev}(-_2)) \parallel \mathsf{devs}(-_1) \tag{5.1}$$

$$\mathsf{loc}(-_0 \mid \mathsf{dev}(-_2)) \parallel \mathsf{devs}(-_1) \rightarrow \mathsf{loc}(-_0) \parallel \mathsf{devs}(-_1 \mid \mathsf{dev}(-_2)) \tag{5.2}$$

$$\mathsf{loc}(-_0 \mid \mathsf{loc}(-_1 \mid \mathsf{dev}(-_2))) \rightarrow \mathsf{loc}(-_0 \mid \mathsf{loc}(-_1) \mid \mathsf{dev}(-_2)) \tag{5.3}$$

$$\mathsf{loc}(-_0 \mid \mathsf{loc}(-_1) \mid \mathsf{dev}(-_2)) \rightarrow \mathsf{loc}(-_0 \mid \mathsf{loc}(-_1 \mid \mathsf{dev}(-_2))) \tag{5.4}$$

Sorts:
$(5.1) : \mathcal{K}_{\mathbf{C_X}}, \mathcal{K}_{\mathbf{C_X}}$
$(5.2) : \mathcal{K}_{\mathbf{C_X}}, \mathcal{K}_{\mathbf{C_X}}$
$(5.3) : \mathcal{K}_{\mathbf{C_X}}$
$(5.4) : \mathcal{K}_{\mathbf{C_X}}$

Figure 5.3: Part $\mathbf{C_X}$ of the Plato-graphical model $\mathcal{X}$, with sorts.

of movement having chosen a tree structure as location hierarchy. One may wonder why we do not simply have one rule to move a device from one location to any other since we do not impose any restrictions on movement. There are two reasons for that: (1) It contradicts the choice of a tree structure of locations, and (2) we would need two rules. We believe the first point has been covered already and proceed to justify the second. Consider the following rule.

$$\mathsf{loc}(-_0 \mid \mathsf{dev}(-_1)) \parallel \mathsf{loc}(-_2) \rightarrow \mathsf{loc}(-_0) \parallel \mathsf{loc}(-_2 \mid \mathsf{dev}(-_1))$$

This wide rule is not what we want. It can, as expected, perform the following reaction:

$$\mathsf{loc}(-_0 \mid \mathsf{loc}(\mathsf{dev}(-_1))) \mid \mathsf{loc}(-_2) \rightarrow \mathsf{loc}(-_0 \mid \mathsf{loc}()) \mid \mathsf{loc}(-_2 \mid \mathsf{dev}(-_1))$$

More generally, it can move a device from one location into another location provided that there is a context which assigns them a *common parent*, which is not one of the locations in question. However, it can *not* perform the following reaction:

$$B = \mathsf{loc}(-_0 \mid \mathsf{dev}(-_1) \mid \mathsf{loc}(-_2)) \rightarrow \mathsf{loc}(-_0 \mid \mathsf{loc}(-_2 \mid \mathsf{dev}(-_1))) = B' \quad \not\downarrow$$

That is, it can not move a device from a location l_1 to a sub-location of l_1. To realise this let us try to construct a match; it suffices to argue that for all C we have $B \neq C \circ (R \oplus \omega) \circ a$:

$$\begin{aligned} R &= \mathsf{loc}(-_0 \mid \mathsf{dev}(-_1) \parallel \mathsf{loc}(-_2)) : (\emptyset, \emptyset, \emptyset) \to (\emptyset, \emptyset) \\ a &= \mathsf{i1} \parallel \mathsf{i2} \parallel \mathsf{i3} : (\emptyset, \emptyset, \emptyset) \\ \omega &= (\mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset) \parallel \mathsf{id}_\emptyset : (\emptyset, \emptyset, \emptyset) \to (\emptyset, \emptyset) \\ C &= \mathsf{id}_\emptyset \mid \mathsf{id}_\emptyset : (\emptyset, \emptyset) \to \emptyset \end{aligned}$$

Clearly, this is not a match, and can not be made so. The intuition is that the context C must have two holes because R has two regions. Intuitively, the trouble is that the context can not take something from one of these holes and put it into the other – it sees the regions of R (and R') from above, so to speak. One could try to circumvent this mechanism by including one location in the other through a parameter a, but that also fails, because the context can not get rid of either one of the parameters it absorbs, and will thus have one location too many for the match. Thus, for the scheme with the wide rule to work as intended, we must include another rule that can move a device into a sub-location. We have such a rule above, namely (5.4) which can be applied consecutively. We conclude that the two movement rules chosen, (5.3) and (5.4), are the better choice.

With this analysis in mind, we remark that the suggested rule (3.10) of Chapter 3 perhaps should be replaced by two rules.

This concludes our treatment of $\mathbf{C_X}$. We proceed with $\mathbf{S_X}$ and then $\mathbf{L_X}$ before gluing together $\mathbf{C_X}$ and $\mathbf{L_X}$ via $\mathbf{S_X}$.

The sensor part $\mathbf{S_X}$

$\mathbf{S_X}$ is the representation of a simple sensor system that can (1) observe (sense) that a device is in a certain location (in $\mathbf{c}_X$), and (2) observe that a device is not located (sensed), i.e., residing in "location" devs. There could be (at least) two reasons for losing track of a device: (1) It was turned off, and (2) the positioning system simply did not pick up on the signal from the device (for a certain amount of time). Recall that the set of devices is fixed and known. $\mathbf{S_X}$ informs $\mathbf{L_X}$ about these observations by invoking certain "interface functions" provided by L_X, where L_X is the MiniML representation of $\mathbf{L_X}$. We use similar notation for A_X. We will examine this communication in Section 5.2.3.

Using the terminology of [RCS06], in our work we assume that sensors are *active* meaning that they observe the world and then react to changes

herein. They do not just sit idle by and let other system parts change their state, which would be *passive*.

Because we have chosen a rather tight coupling between $\mathbf{c}_X$ and $\mathbf{l}_X$, namely to represent location and device identifiers by the same natural numbers in both worlds, $\mathbf{s}_X$ does not need to maintain a mapping from one world to the other. We do require that Ξ-integers are represented in the same way as integers in $\mathbf{C_X}$ – namely by i-controls. $\mathbf{S_X}$ is written in Bigraphs, but also has access to bigraphical *representations* of Ξ- terms.

The signature and dynamics of $\mathbf{S_X}$ are defined as follows in Figure 5.4; sensor $\mathbf{S_X} = (\mathcal{K}_{\mathbf{S_X}}, \mathcal{R}_{\mathbf{S_X}})$.

Rule (5.5) models the case where $\mathbf{s}_X$ observes a device in a location, and informs L_x of this. This is done by "calling" the function in $\mathbf{l}_X$, exported by g as the first component in a tuple, with the bigraphical representations of the device and the location. invoke models a pool of pending function calls and must have name $funs$ and be empty in the initial configuration of $\mathbf{s}_X$. In the rule we have that var_g refers to exactly the same name as invoke_g to force the var-control to refer to precisely the name exported from L_x. To fire, the context of the rule must identify g and $funs$. We need to encapsulate the location identifier in a id-control to distinguish it from sub-locations and thus make the rule work for any location (identifier) $-_0$. Rule (5.6) applies when $\mathbf{S_X}$ observes a device inside the devs control. We will show the relevant functions in Section 5.2.3.

Notice how $\mathbf{S_X}$ merely observes $\mathbf{c}_X$ (sensing) and informs $\mathbf{L_X}$ (acquisition). This may very well lead to discrepancy between $\mathbf{c}_X$ and $\mathbf{l}_X$. To make the system more precise, i.e., to ensure a tighter correspondence between $\mathbf{c}_X$ and $\mathbf{l}_X$ one could allow $\mathbf{S_X}$ to also observe $\mathbf{l}_X$, and then only inform $\mathbf{L_X}$ of the location of a device when $\mathbf{c}_X$ and $\mathbf{l}_X$ disagree. This is not the way positioning systems work in reality, but may be useful for simulation purposes. For now, we leave $\mathbf{S_X}$ as is. This concludes our treatment of the positioning system $\mathbf{S_X}$.

The location model part $\mathbf{L_X}$

This part is implemented in Ξ_{sugar} because writing it natively in Bigraphs proved too cumbersome. We saw the "findall" query in Section 3.A of Chapter 3, and it is significantly more involving to encode more advanced queries. Here we explain the ideas involved in this part of the model, and in Section 5.2.4 we explain how to go from a Ξ_{sugar}-program to a Ξ-program automatically. The presentation of L_X is divided into the following parts:

Sensor $\mathbf{S_X}$.

Control	Activity	Arity	Comment
fst	active	$0 \to 0$	First part of pair
snd	active	$0 \to 0$	Second part of pair
app	active	$0 \to 0$	Application
appl	active	$0 \to 0$	Left part of application
appr	active	$0 \to 0$	Right part of application
var	atomic	$0 \to 1$	Variable
exp	passive	$0 \to 0$	Delay evaluation
i0,i1,i2 ...	atomic	$0 \to 0$	Infinite family of identifiers
invoke	active	$0 \to 1$	Hosts "function calls"

$$\begin{aligned}
&\mathsf{loc}(\mathsf{id}(-_0) \mid \mathsf{dev}(-_1) \mid -_2) \parallel \mathsf{invoke}_g(-_3) \\
&\quad \rightarrowtriangle \\
&\mathsf{loc}(\mathsf{id}(-_0) \mid \mathsf{dev}(-_1) \mid -_2) \parallel \\
&\mathsf{invoke}_g(-_3 \mid \\
&\qquad \mathsf{app}(\mathsf{appl}(\mathsf{app}(\mathsf{appl}(\mathsf{fst}(\mathsf{var}_g)) \mid \\
&\qquad\qquad\qquad \mathsf{appr}(\mathsf{exp}(-_1)))) \mid \\
&\qquad\quad \mathsf{appr}(\mathsf{exp}(-_0))))
\end{aligned} \tag{5.5}$$

$$\begin{aligned}
&\mathsf{devs}(-_0 \mid \mathsf{dev}(-_1)) \parallel \mathsf{invoke}_g(-_2) \\
&\quad \rightarrowtriangle \\
&\mathsf{devs}(-_0 \mid \mathsf{dev}(-_1)) \parallel \\
&\quad \mathsf{invoke}_g(-_2 \mid \mathsf{app}(\mathsf{appl}(\mathsf{fst}(\mathsf{snd}(\mathsf{var}_g))) \mid \mathsf{appr}(\mathsf{exp}(-_1))))
\end{aligned} \tag{5.6}$$

Sorts:
(5.5) : $\mathcal{K}_{\mathbf{C_X}}, \mathcal{K}_{\mathbf{S_X}}$
(5.6) : $\mathcal{K}_{\mathbf{C_X}}, \mathcal{K}_{\mathbf{S_X}}$

Figure 5.4: Part $\mathbf{S_X}$ of the Plato-graphical model $\mathcal{X}$, with sorts.

- Data-type declarations.
- The building configuration.
- The interface to $\mathbf{S_X}$; reconfigurations.
- The interface to $\mathbf{A_X}$; location-based queries.
- Communication between $\mathbf{L_X}$ and $\mathbf{S_X}$.

When these pieces are in place we glue them together to form a presentation of (the structure of) L_x as a whole. To keep the discussion focused we take the liberty of being very brief when treating auxiliary functions in A_x – we are interested in the reconfigurations of $\mathbf{l}_X$ and the queries supplied to $\mathbf{A_X}$.

Data-type declarations are a convenient way to abstract away from the underlying type when programming in (a fraction of) SML. In L_x we choose to work with just one basic data-type for location and device identifiers; natural numbers. Natural numbers and equality on them can be easily encoded in Bigraphs, as seen in Chapter 4. We could have used strings instead, but natural numbers are simpler and suffice for our purposes.

The data-types used in L_x are as follows.

```
type lid = int
type dev = int
datatype hierarchy = (* id, devices, sublocations *)
         Loc of lid * dev list * hierarchy list
```

A hierarchy represents the aforementioned location tree. A location has an identifier, a list of devices, and a list of sub-locations. Both types and data-types belong to the set C of constructors. For now, we assume lists as a predefined data-type. We also assume a few basic operations on lists, namely cons ('::'), append ('@'), reverse ('rev'), and 'map'. We address how to encode lists and the associated operations in Ξ in Section 5.2.4.

The building configuration The building looks like this in Ξ_{sugar} and corresponds exactly to the one in $\mathbf{C_X}$. The initial configuration $\mathbf{l}_X$ is shown below along with some enclosing Ξ_{sugar} code.

```
val funs =
  let val state =
    ref (Loc(1,[15],
             [Loc(2,[10,11],[]),
              Loc(3,[],[]),
              Loc(4,[],
                  [Loc(5,[12],[]),
                   Loc(6,[],
                       [Loc(7,[],
                            [Loc(8,[13],[]),
                             Loc(9,[14],[])])])])])))
      val devs = ref []
```

```
    ...
in ... end
```

As can be seen, we implement devs as (a reference to) a list. We use references to update the internal representation of $\mathbf{l}_X$. The Ξ-locations are translated into Bigraphs by the translation given for constructors in Chapter 4. The dynamics of L_x, i.e., reconfigurations of this configuration are implemented by Ξ-functions, to which we will return later. Three dots signify place-holders, ignore them for now. We explain what happens to the 'val' declaration in Section 5.2.4.

The interface to $\mathbf{S_X}$; reconfigurations $\mathbf{L_X}$ supplies $\mathbf{S_X}$ with some interface functions. The purpose of these functions is to allow $\mathbf{S_X}$ to inform $\mathbf{L_X}$ of events (movement of devices), which $\mathbf{S_X}$ has observed in $\mathbf{C_X}$. As mentioned, $\mathbf{S_X}$ can do two things:

- Observe that a device is in a certain location, and inform $\mathbf{L_X}$.
- Observe that a device is not sensed in any location, i.e., currently not in use and residing in "location" devs, and inform $\mathbf{L_X}$.

These two situations give rise to two reconfigurations, i.e., Ξ_{sugar}-functions.

```
fun sobserved d =
    fn l =>
       let val state' = delete d (!state)
           val devs' = del_list d (!devs)
           val state'' = insert d l (!state)
       in state:=state''; devs:=devs' end

fun slost d =
    let val state' = delete d (!state)
        val devs' = del_list d (!devs)
        val devs'' = d::devs'
    in state:=state'; devs:=devs'' end
```

The function sobserved is called by $\mathbf{S_X}$ (we will return to what a "function call" means) when $\mathbf{S_X}$ observes that device d is in location l in $\mathbf{c}_X$. The function makes sure to delete d everywhere from the state hierarchy $\mathbf{l}_X$, and then inserts it in the (possibly) new location l. Like in $\mathbf{c}_X$ this guarantees that d is either in exactly one location or in devs at any point in

time. This is under the assumption that it is called with an existing location. The function `slost` is similar, but it places `d` in `devs`. The auxiliary functions `delete`, `del_list`, and `insert` are functions internal to $\mathbf{l}_X$, and `state` and `devs` are two private data structures in $\mathbf{l}_X$ constituting the current configuration. We show these functions below for completeness.

```
(* remove an element from a list *)
fun del_list e =
    fn [] => []
     | (x::xs) => if e=x then del_list e xs
                  else x :: del_list e xs

(* delete device 'dev' from hierarchy 'id' *)
fun delete dev =
    fn (Loc(id,ds,ls)) =>
       Loc(id, del_list dev ds, map (delete dev) ls)

(* insert device 'dev' into location 'lname'
   in hierarchy 'id' *)
fun insert dev =
    fn lname =>
       fn (Loc(id,ds,ls)) =>
          if lname=id then Loc(id,
                               dev::ds,
                               map (insert dev lname) ls)
          else Loc(id, ds, map (insert dev lname) ls)
```

They should be straightforward and the comments sufficient. We do, however, note that the functions are curried and thus take exactly one parameter. Also, notice how $\mathbf{s}_X$ does not know the internals of $\mathbf{l}_X$, but merely the names of the two interface functions 'sobserved' and 'slost'.

The interface to $\mathbf{A}_X$; location-based queries The interface to $\mathbf{A}_X$ is currently defined by two functions visible to $\mathbf{A}_X$, but with internal workings local to $\mathbf{L}_X$. It is possible for the application ($\mathbf{A}_X$) to enquire to the whereabouts of a certain device, and to enquire about all the devices to be found in a certain location.

```
fun awher d = whr d (!state)
fun afind lname = flocs (pickloc lname (!state))
```

These functions have no side effects, in particular they do not alter $\mathbf{1}_X$. The function 'awher' is based on the following auxiliary functions, which should be read bottom-up.

```
(* find the identifier of a device's location *)
fun whr dev =
    fn l =>
       case l of
           (Loc(_,[],[])) => NONE
         | (Loc(id,d::ds,ls)) =>
           if dev=d then SOME(id)
           else whr dev (Loc(id,ds,ls))
         | (Loc(_,[],ls)) =>
           let fun whr' =
                   fn list =>
                      case list of
                         [] => NONE
                       | (loc::locs) =>
                            case whr dev loc of
                               SOME(i) => SOME(i)
                             | NONE => whr' locs
           in whr' ls end
```

We believe that the comments are sufficient explanation. We address 'NONE', 'SOME(x)' (which can also be used in $\mathbf{A}_X$) and embedded functions in Section 5.2.4. The function 'afind' uses the following auxiliary functions.

```
(* find all devices in a hierarchy - depth first *)
fun fall (Loc(_,ds,[])) = ds
  | fall (Loc(_,ds,l::ls)) =
    let fun fall' [] = []
          | fall' (loc::locs) = (fall loc) @ (fall' locs)
    in ds @ (fall l) @ (fall' ls) end

(* pick the subtree with id 'loc' from a hierarchy *)
fun pickloc lname =
    fn (Loc(id,ds,ls)) =>
       if lname=id then SOME(Loc(id,ds,ls))
       else let fun pickloc' [] = NONE
                  | pickloc' (loc::locs) =
```

```
                    case pickloc lname loc of
                        SOME(l) => SOME(l)
                      | NONE => pickloc' locs
            in pickloc' ls end

(* unpack option, return list of devices *)
fun flocs option =
    case option of NONE => []
                 | SOME(l) => fall l
```

Again, we refer to the comments for explanation.

Communication between $\mathbf{L_X}$ and $\mathbf{S_X}$ We promised to address the question of what it means to call a Ξ_{sugar}-function (in this case living in $\mathbf{L_X}$) from a native BRS (in this case $\mathbf{S_X}$). We need to be able to somehow export selected function names from $\mathbf{L_X}$ to $\mathbf{S_X}$. Such a mechanism is not present in Ξ, so we introduce it by means of additional syntax. The effect is that L_x has the following form:

```
export <name> from <exp>
```

The idea is to export the names of some functions from L_x to other parts of the system. One can think of it as a special form of `val f = exp` known from SML. This is done by making a particular name global in the Plato-graphical system (by closure), and then to make sure that the translation of relevant functions in 'exp' in L_x refer to this name. The translation of this new syntactic construction is:

$$[\![\text{export } f \text{ from } exp]\!]_X = \mathsf{def}_f([\![exp]\!]_\emptyset)$$

We emphasise the fact that we export one name f, which can possibly name a tuple of function names (as is the case for $\mathbf{l}_X$). Thus, we can use projections on the name ('val') f to refer to the individual function names of the tuple (nested pairs). This was done in $\mathbf{S_X}$ where we used an explicit projection on a name g that must match the exported name f.

The Plato-graphical system with function export has the form

$$/f.\mathbf{C_X} \parallel \mathbf{S_X} \parallel \mathsf{def}_f([\![exp]\!]_\emptyset) \parallel [\![A_X]\!]_f$$

where A_X is the component $\mathbf{A_X}$ before translation into Bigraphs. Notice that exp is translated from Ξ_{sugar} into Bigraphs using an empty set. This ensures

that $f \notin \mathsf{fv}(exp)$. f will be referred to from $\mathbf{S_X}$. For example, f could be `funs` and export a tuple of function names `sobserved` and `slost`.

It is important to be aware of the fact that $\mathbf{S_X}$ accesses the function 'sobserved' etc. by linking controls (vars in this case) to the projections on the exported name f. As we saw in the treatment of $\mathbf{S_X}$ above, $\mathbf{S_X}$ can produce function calls in $\mathbf{L_X}$ by introducing applications of var controls to (bigraphical representations of) arguments. We presented $\mathbf{L_X}$ in a form where functions take exactly one argument to match the way $\mathbf{S_X}$ introduces function calls.

To sum up, L_x has the following form:

```
export funs from
   let val ...
       fun sobserved d = fn l => ...
       fun slost d = ...
       ...
   in (sobserved, slost, ...) end
```

This concludes our treatment of inter-component communication between $\mathbf{S_X}$ and $\mathbf{L_X}$.

$\mathbf{L_X}$ collected We present the structure of L_x as a whole, but we leave out (denoted by three dots) the auxiliary functions and comments, for readability.

```
export funs from
  type lid = int
  type dev = int
  datatype hierarchy =
           Loc of lid * dev list * hierarchy list
  ...
  let val state =
        ref(Loc(1,[15],
                [Loc(2,[10,11],[]),
                 Loc(3,[],[]),
                 Loc(4,[],
                     [Loc(5,[12],[]),
                      Loc(6,[],
                          [Loc(7,[],
                               [Loc(8,[13],[]),
```

```
                                Loc(9,[14],[])])])])]))
    val devs = ref []
    fun sobserved d =
        fn l =>
          let val state' = delete d (!state)
              val devs' = del_list d (!devs)
              val state'' = insert d l (!state)
          in state:=state''; devs:=devs' end
    fun slost d =
          let val state' = delete d (!state)
              val devs' = del_list d (!devs)
              val devs'' = d::devs'
          in state:=state'; devs:=devs'' end
    fun awher d = whr d (!state)
    fun afind lname = flocs (pickloc lname (!state))
in (sobserved,slost,awher,afind) end
```

The signature $\mathcal{K}_{\mathbf{L_X}}$ for $\mathbf{L_X}$

Apart from the controls given in Figure 4.1 we obtain the following controls by the translation given in Chapter 4. They are shown in Figure 5.5.

Model $\mathbf{L_X}$.

Control	Activity	Arity	Comment
Loc	active	$0 \to 0$	Location hierarchy constructor
Nil	passive	$0 \to 0$	Empty list constructor
Cons	active	$0 \to 0$	List constructor
NONE	passive	$0 \to 0$	Empty option constructor
SOME	active	$0 \to 0$	Option constructor

Figure 5.5: Additional controls for the signature of $\mathbf{L_X}$.

This concludes our treatment of $\mathbf{L_X}$.

Dynamic correspondence between $\mathbf{C_X}$ and L_x

$\mathbf{C_X}$ and L_X uphold the same invariants:

- A device is either in devs or exactly one location at any time.
- Locations and devices are uniquely defined, and in one-to-one correspondence between $\mathbf{C_X}$ and L_X.

We remark that it is reasonable to extend L_x with a parent map without doing so in $\mathbf{C_X}$ because the internal representation and data structures in L_x are of no concern to the model of the real world as long as the invariants are kept. We stress the fact that these two worlds must agree on the representation of location identifiers, that is, natural numbers. $\mathbf{C_X}$ uses i−controls, and so does $\mathbf{L_X}$ because of the translation defined. It is, however, up to the specifier to make sure that the same numbers are given to corresponding locations in both worlds.

The application part $\mathbf{A_X}$

This part should be a simple Ξ_{sugar}-program that uses the queries supplied by L_x. An example application could be "find the nearest printer" with respect to the current location of my mobile device. We do not actually give such a program here, but merely state that such a program should be straightforward to construct given what we have done in L_X. For an example of an interesting real-life location-aware application we refer the reader to the implementation of the Lancaster tour guide GUIDE of Appendix A.2.2, which is discussed in Section 6.4.

5.2.4 *From Ξ_{sugar} to Ξ*

Ξ_{sugar} is a sugared version of Ξ which additionally has a mechanism to export function names to top level of Plato-graphical systems, as explained earlier. Most constructs in Ξ_{sugar} can simply be unfolded to or encoded as Ξ-constructs, but we also have the name-exporting enhancement adding modelling power to the calculus.

We begin with the enhancement. This is the main difference between Ξ_{sugar} and Ξ. The effect is that Ξ_{sugar}-programs p' are not just expressions, but are now encapsulated:

$$p' ::= \text{export } f \text{ from } p$$

The rest of the constructions are syntactic sugar. They are:

- Comments.
- 'type' and 'val' constructs.
- Nested 'let' construct.
- Tuples.

- Anonymous and nominal function declarations.
- 'if-then-else' conditionals.
- Basic operations on natural numbers.
- Patterns.

We treat each item in turn.

Comments are enclosed by '(*' and '*)', and are simply discarded when translating L_x into Bigraphs.

'type' and 'val' are handled as follows. Type declarations are treated as textual substitution on the source program such that `type dev = int` is discarded and `dev` is replaced by `int` everywhere in the source program, for example. This, of course, requires that 'dev' is not used as a variable name, i.e., $\mathcal{V} \cap C = \emptyset$, as mentioned earlier. This means that any constructor name from C is legal in a Ξ_{sugar}-program, and that it is the programmer's responsibility to use the so constructed terms correctly, i.e., to make sure that case expressions on a custom constructor have the right number of branches corresponding to the constructor.

Let us consider Booleans and lists to see how the general constructor and case terms of Ξ_{sugar} can be represented in Ξ. We would want to introduce abbreviations for null-ary constructors; 'True', 'False', and 'Nil'. For Booleans the following terms are legal:

- 'True unit',
- 'False unit', and
- 'case e of True $x_1 \Rightarrow e_1$ | False $x_2 \Rightarrow e_2$'.

For lists:

- 'Nil unit',
- 'Cons (x_1, x_2)', and
- 'case e of Nil $x \Rightarrow e_1$ | Cons $x \Rightarrow$ let h = fst x in let t = snd x in e_2 end end'.

We could also introduce syntactic sugar for the 'case' construct on lists:

$$\text{'case } e \text{ of Nil} \Rightarrow e_1 \mid \text{Cons } (h, t) \Rightarrow e_2' \ .$$

As an example we show the semantics of lists (in Ξ and Ξ_{sugar}), which are more interesting than for Booleans, would then be (leaving out the store since there is no side-effect):

$$\begin{aligned}
&\text{case Nil of Nil} \Rightarrow e_1 \\
&\qquad \mid \text{Cons } x \Rightarrow \text{let } h = \text{fst } x \text{ in let } t = \text{snd } x \text{ in } e_2 \text{ end end} \\
&\qquad \longrightarrow e_1 \\
&\text{case Cons } (v_1, v_2) \text{ of Nil} \Rightarrow e_1 \\
&\qquad \mid \text{Cons } x \Rightarrow \text{let } h = \text{fst } x \text{ in let } t = \text{snd } x \text{ in } e_2 \text{ end end} \\
&\qquad \longrightarrow (\text{let } h = \text{fst } x \text{ in let } t = \text{snd } x \text{ in } e_2 \text{ end end})\{(v_1, v_2)/x\}
\end{aligned}$$

The operations (functions) in lists, namely '::', '@', 'rev', and 'map' can clearly be coded in Ξ_{sugar} (and Ξ). The two functions '::' and '@' are used infix, but can easily be made prefix.

We have just seen how to represent Booleans and lists. Another used data-type in L_X is options, that is, 'NONE' and 'SOME(x)'. Having presented lists we trust that the reader is convinced that options can be represented similarly. A 'val' declaration inside a 'let' expression is discarded, so

```
let val x = exp1 in exp2 end
```

becomes

```
let x = exp1 in exp2 end .
```

Nested 'let' constructs can simply be unfolded. Thus,

```
let val x = ...
    val y = ...
in ... end
```

becomes

```
let val x = ... in
  let val y = ... in ... end
end
```

where the 'val' and 'end' keywords are discarded under translation.

Tuples are simply nested pairs such that $(a, b, c) = (a, (b, c))$ and so forth.

Anonymous and nominal function declarations We use two different ways of declaring functions in L_X; anonymous and nominal. The anonymous functions are of form `fn x => e` and are translated into '$\lambda x.e$'. Nominal functions on form '`fun f x = e ...`', where the dots signify the rest of the program, are translated into '`let f = (fix f(x) = e) in ...`', because they can be recursive.

'if-then-else' conditionals are simply unfolded using the 'case' construct so '`if b then e1 else e2`' becomes '`case b of True => e1 | False => e2`'.

Basic operations on natural numbers are assumed to exist as primitives in Ξ_{sugar} because we have already shown that we can encode them in Bigraphs. The operations used are '=', '<=' (less than or equal to), and '-' (subtraction) on natural numbers.

Patterns are used heavily in L_X. It is perhaps easiest to see how patterns are unfolded by seeing and example. Consider the function `del_list` shown above:

```
fun del_list e =
    fn [] => []
     | (x::xs) => if e=x then del_list e xs
                  else x :: del_list e xs
```

First, unfold the 'fun':

```
val del_listl = fix del_list(e) =
    fn [] => []
     | (x::xs) => if e=x then del_list e xs
                  else x :: del_list e xs
```

Then, unfold the pattern:

```
val del_listl = fix del_list(e) =
    lambda y. case y of [] => []
                   | Cons z => let x = fst z in
                               let xs = snd z in
                               if e=x then del_list e xs
                               else x :: del_list e xs
```

where 'lambda' signifies λ. This Ξ_{sugar}-expression can be translated into a Ξ-expression by the methods shown above.

There are also more advanced patterns in use, i.e., patterns where we do not match merely a constructor like 'List', but a composite constructor such as 'Loc(5,Nil,Nil)'. Consider this case:

```
case exp of Loc(5,Nil,Nil) => e
```

We unfold this into

```
case exp of Loc p =>
  let a = fst p in
    let b = fst (snd p) in
      let c = snd (snd p) in e
      end
    end
  end
```

adhering to the restriction that the left-hand side of a 'case' branch must consist of a constructor and a variable.

This concludes our justification of the claim that Ξ_{sugar}-programs can be translated into Local Bigraphs, via Ξ.

5.2.5 *The model $\mathcal{X}$ is Plato-graphical*

Having explained the model we state and prove that it is actually Plato-graphical. This proposition relies on the fact that a given implementation of $\mathbf{A}_\mathbf{X}$ uses a subset of the controls of the $\mathbf{L}_\mathbf{X}$ as given above.

Proposition 5.1. *The model $\mathcal{X}$ is Plato-graphical.*

Proof. It is enough to observe that $\mathbf{C}_\mathbf{X} \perp \mathbf{A}_\mathbf{X}$, and that $\vec{x} = funs$.

This concludes our presentation of the bigraphical location model. We proceed by relating the presented model to those of Chapter 2.

5.2.6 *Relating our model to the location model classification*

In Chapter 2 we introduced and classified location models. We justify our claim that our model is representative for a class of the presented models presented in Chapter 2. It is important for our model to be representative because that property renders our work relevant for a wide range of location models, and thus a wide range of location systems.

Classifying the model

Clearly, the model $\mathcal{X}$ is symbolic, but not geometric. Thus, there is no notion of metric distance. We have shown that the location model L_x supports location queries and certain reconfigurations, and also the accumulating "find all" query, among others. We have not shown how to support range and navigation queries, but these queries can be easily programmed by introducing an explicit parent map to the location hierarchy. A range query is then a little primitive in the sense that ranges are determined by location containment. One could consider adding weights to edges and thereby enable shortest-path navigation queries. We have implemented these queries, but do not show them here because they do not add anything conceptual to this treatment. Some non-trivial work still remains to support "nearest neighbour" queries, namely to instrument the model with geometric information. To this end we could perhaps instrument each location with a coordinate control holding an ordered triple of integers.

We propose to look at the location hierarchy and queries supported for deciding when a given model correctly implements a specification. We conclude that the model is representative in the sense that it captures the essentials of an exclusive symbolic location model.

5.3 Concluding remarks

First, we consider whether we are any closer to answering the five questions of Chapter 3:

1. What languages $\mathcal{L}$ can we encode?
2. How close are Plato-graphical models to real systems?
3. What challenges have we found for bigraphical models?
4. What uses do we envision for Plato-graphical models?
5. How do we reason about Plato-graphical models?

Ad. 1. We have successfully encoded an extended version of MiniML, which should to enable us to encode a wide range of location-aware applications. It turns out that reaction rules are enough for $\mathbf{C}_\mathbf{X}$ (and $\mathbf{S}_\mathbf{X}$).

Ad. 2. We have enough structure to represent an exclusive, symbolic location model. We have found possible uses for DAGs, timed and probabilistic events, and continuous space. Such extensions can be used to lift the model to real-life systems.

Ad. 3. We found that one may use closed links in a clever way (see Chapter 3) to handle that something is *not* present in the context under reaction. We do, however, envision that this so-called "negative" context information will be needed in the bigraph theory in the long run, and conjecture that a safe sorting exists to enforce this.

Ad. 4. We envision to implement Plato-graphical models according to specifications of location models (and also context models) when a tool allows us to perform automated reactions. This is the basis of the simulation challenge.

Ad. 5. This is still an open question. One question is: To what extend can we transfer reasoning about Ξ-programs to Bigraphs? As an example we could mention contextual term equality. That is, if two Ξ-terms f and g are contextually equivalent, then are their images under $[\![\cdot]\!]$ equal? One could argue that this question should be studied in a simpler and better understood framework than Plato-graphical systems. Another questions is: What does it take for us to be able to substitute one component for another in a Plato-graphical system (recall Proposition 3.4.1 and Definition 3.4.4 of Chapter 3)? This is indeed an important question for future work.

5.3.1 Conclusions on our modelling effort

We draw the following conclusions:

- We can encode an extended version of MiniML in (Local) Bigraphs.
- We can represent an exclusive symbolic location model and all the desired query types on it in Bigraphs. (We ask the reader to trust that we can program symbolic range and navigation queries as we have not shown these explicitly.)
- We can represent the world and a simple positioning/sensor system in Bigraphs.
- We have taken one more step in evaluating Plato-graphical systems and thereby Bigraphs as a modelling formalism for GUC.
- We have argued that Plato-graphical systems enable convenient *modelling* of location-aware systems, i.e., facilitate programming of a location-aware application in Ξ_{sugar} querying a location model.
- The modelling effort was not low, but we now have a fairly accurate base system to support many other agents (location-aware applications).

This concludes the chapter on a real-life location model. We proceed with a chapter on simulation to "close the circle".

6

Simulation of Location-aware Systems

In this chapter we present some results of simulation of a location model. Using Plato-graphical models we can simulate not just which output the positioning systems yields, but also the sensor part itself and the following processing in the location model part. There are at least three ways to benefit from a bigraphical (location) model:

- A specification of how a (location) model should behave.
- An implementation of a (location) model.
- A layer of abstraction to facilitate the creation of location-based services.

Describing the intended behaviour formally as a bigraphical model can be advantageous for several reasons. First, it helps to clarify unclear points in the informal specification. Second, in a model one often omits some unimportant details allowing one to distill the essence. Seen as an implementation it is useful because one very often encounters issues when operationalising a (declarative) system specification. Furthermore, when a system is implemented we may actually experiment with it. Finally, when seen as a layer of abstraction, one can simply run the model and try out different LBSs interfacing with it. The model is a dynamic execution environment for location-aware applications; the programmer need not worry about details of the location model while programming the application. The key to exploiting these aspects is simulation, because without computer aid it is infeasible to work with large(r) systems. Furthermore, simulation is a cheap and quick method of prototyping applications and protocols, which will play a key role in development and testing [RCS06].

The core of this chapter is an abstract version of the model presented in the previous chapter. The reason is two-fold:

- It provides a more tangible case study aiding presentation.
- The BPL tool can not currently handle a model of the magnitude of the one in Chapter 5.

We discuss the second point in a little more detail below. The model is abstract in the sense that the operational details are no longer apparent, in particular in the location model part. In fact, we consider two different abstract models. We consider each one in turn.

6.1 An abstract location model

The main change in the model is a simplifaction of **L**, but we have also made some changes to the other parts of the model. We remark that arities are given for Binding Bigraphs, although we use no binding ports in this model, but the implementation is in Binding Bigraphs using the BPL tool. Without further ado we present the particulars of the abstract model.

6.1.1 World, $\mathbf{C}'$

The world part $\mathbf{C}'$ is more or less as before, however, it no longer maintains a list of unused devices in a devs node. Thus, rules to discover and lose devices are no longer needed. There is just a fixed set of devices in the, say, building. Another difference is that identification of locations and devices is now implemented by links. Notice that the sort of each control is $\mathcal{K}_{\mathbf{C}'}$, and that both redex and reactum contain controls of this sort only.

```
signature world =
  sig % arity is: (binding -> free)
    loc : passive (0 -> 1)
    dev : atomic  (0 -> 1)
  end

using world

rule moveup =
  loc_l([0] | loc_l'([1] | dev_d))
    ->
  loc_l([0] | loc_l'([1]) | dev_d)

rule movedown =
```

```
  loc_l([0] | loc_l'([1]) | dev_d)
    ->
  loc_l([0] | loc_l'([1] | dev_d))
```

Devices may move up or down one step in the tree hierarchy. The state of $\mathbf{C}'$ is a location tree (nested loc nodes) with devices embedded.

6.1.2 *Sensor,* $\mathbf{S}'$

The sensor part $\mathbf{S}'$ makes sure that the list of devices and their location, in $\mathbf{L}'$, is updated with knowledge acquired from $\mathbf{C}'$.

```
signature sensor =
  sig % arity is: (binding -> free)
  end

rule observe_update =
  loc_l(dev_d | [0]) || devs(location_l',d | [1])
    ->
  loc_l(dev_d | [0]) || devs(location_l,d  | [1])

rule observe_new =
  ( /d . loc_l(dev_d | [0]) ) || devs([1])
    ->
  /d. ( loc_l(dev_d | [0]) || devs(location_l,d | [1]) )

rule lose =
  loc_l([0]) || ( /d. devs(location_l',d | [1]) )
    ->
  loc_l([0]) || devs([1])
```

The signature of $\mathbf{S}'$ is empty because the sensor's only functionality is to observe $\mathbf{C}'$ and communicate what it sees to $\mathbf{L}'$. It only uses their controls, as it has no state by itself.

There are three reaction rules. As before, it can still observe and lose devices, however, there is no invoke node because in the abstract model there is no need to control function calls and buffers with spin-locks and so forth, because all operations are atomic in the sense that they are implemented by a single reaction rule. The sort of each rule is $\mathcal{K}_{\mathbf{C}'}, \mathcal{K}_{\mathbf{L}'}$. This is in harmony with the fact that $\mathbf{S}'$ observes $\mathbf{C}'$ and $\mathbf{L}'$, and informs the latter if a discrepancy is discovered.

The rule for updating can fire whenever a device is in another location in $\mathbf{C}'$ than in $\mathbf{L}'$. The result is that $\mathbf{L}'$ has its state updated directly by the sensor part with the current world location of the device, and the obsolete information is removed. If a device is seen in $\mathbf{C}'$ but does not appear in the state of $\mathbf{L}'$ (if the device identifier d is closed) then $\mathbf{S}'$ simply adds it. If a device is known in $\mathbf{L}'$ but not in $\mathbf{C}'$ then it is removed from $\mathbf{L}'$.

6.1.3 Location model, $\mathbf{L}'$

The abstraction consists mainly in simplifying $\mathbf{L}'$, both because this is the model part causing the problems with magnitude for the tool, but also because we do not lose much information by the simplification, and the overall picture becomes clearer. There is no great necessity that $\mathbf{L}'$ should represent $\mathbf{C}'$ very accurately. The abstract location model part, $\mathbf{L}'$, merely maintains a collection of observations about where devices were last seen. This limits which queries can be supported, but still we may ask "where is device d" and "find all devices".

Having everything expressed natively in Bigraphs, with no MiniML code, simplifies things greatly. This, in itself, may be useful with respect to formal reasoning.

The devs node simply holds all the location nodes, and each of these represents that a particular device (identified by a link) is at a particular location (also identified by a link). Both controls are of sort $\mathcal{K}_{\mathbf{L}'}$.

```
signature repr =
  sig % arity is: (binding -> free)
    devs     : passive (0 -> 0)
    location : atomic (0 -> 2) % linked to loc and dev
  end
```

Evidently, there are no rules in $\mathbf{L}'$. The state of $\mathbf{L}'$ will be of form

$$\mathsf{devs}(\mathsf{location}_{l_1,d_1} \mid \ldots \mid \mathsf{location}_{l_n,d_n}) \,,$$

so device d_i is in location l_i.

Because the representation in $\mathbf{L}'$ does not reflect the complete hierarchy there is no reason to require or formulate that $\mathbf{C}'$ and $\mathbf{L}'$ should agree on the initial state of the world.

6.1.4 Application, A′

The application part **A′** can perform "where is" and "find all" queries. Finding all devices is simple as it is just the content of the devs node of **L′**. "Where is" looks in the list of devices maintained by **L′**, which is easy because we now have an identifying link, cf. the modificaton of **C′**.

```
signature agent =
  sig % arity is: (binding -> free)
    findall  : atomic (0 -> 0)
    whereis  : atomic (0 -> 1) % linked to the device sought
  end

using agent

rule findall =
  devs([0]) || findall
    ->
  devs([0]) || [0]

rule whereis =
  devs(location_l,d | [0]) || whereis_d
    ->
  devs(location_l,d | [0]) || location_l,d
```

The two queries in question are represented by corresponding controls. Two rules implement their functionality. Both rules have sort $\mathcal{K}_{\mathbf{L}'}, \mathcal{K}_{\mathbf{A}'}$. They observe the state of **L′** and directly report the finding. If a "find all" query is present, by means of a findall node, then all devices known to **L′** at that point in time, are returned. "Where is" simply looks in **L′** to return the location of the device in question.

This concludes the presentation of the abstract model. The reader will hopefully agree that the model is simple, and that it is a reasonable abstraction to work with.

6.2 A pedagogical scenario

In this section we take the reader through an example, the purpose of which is to unveil some minor subtleties in the abstract model. The reader may already have discovered the forthcoming issues, but mind you that this is only a simple example – there is no guarantee that one would realise these

points at the time of design of the model (or system), let alone *all* possible potential issues.

Consider the following initial state of our abstract model, a Plato-graphical system, where we use non-ASCII notation:

$$\mathbf{c}' = \mathsf{loc}_{l'}(\mathsf{loc}_{l}(\mathsf{dev}_{d})) \qquad \mathbf{l}' = \mathsf{devs}(\mathsf{location}_{l',d}) \qquad \mathbf{a}' = \mathsf{whereis}$$

The scenario is as follows:

1. A "whereis" query for device d is issued.
2. A move of d occurs in $\mathbf{c}'$.
3. An answer to the query appears in $\mathbf{a}'$.
4. Another "whereis" query is issued, this time for device d'.
5. $\mathbf{S}'$ discovers that d' occurs in $\mathbf{c}'$ but not in $\mathbf{L}'$ and reacts.

This scenario is rather simple. Nevertheless, it reveals some points that will be discussed as we progress with the system's evolution.

First, what does it mean to "issue a query"? Well, we chose that vague formulation on purpose to illustrate that a decision has to be taken on whether queries are just in $\mathbf{a}'$ by initialisation, or whether they are generated as the system evolves. Let us pick the second solution, as this is most realistic. Thus, we add two rules to $\mathbf{A}'$ to generate queries, one for each. They look as follows, again using non-ASCII notation:

$$-_0 \multimap -_0 \mid \mathsf{findall} \qquad \text{and} \qquad -_0 \mid (d/ \otimes 1) \multimap -_0 \mid \mathsf{whereis}_d \ .$$

This is not extremely elegant as the idle name clutters the presentation. Nevertheless, it is necessary because of the condition $cod(R) = cod(R')$ for reaction rules. Do notice that we do *not* have to initialise the $\mathbf{A}'$ part of the system with idle names corresponding to the names of all the devices in $\mathbf{c}'$, because this idle name can be added by the context for matching purposes.

Now, assuming that a "whereis d" query has been generated in $\mathbf{A}'$, we may continue with our scenario. The next event is that d moves up in $\mathbf{C}'$ yielding a new state:

$$\mathbf{c}' = \mathsf{loc}_{l'}(\mathsf{loc}_{l}() \mid \mathsf{dev}_{d}) \ .$$

Suppose that rule `whereis` fires in $\mathbf{A}'$. That changes the state of $\mathbf{A}'$

$$\text{from} \qquad \mathbf{a}' = \mathsf{whereis}_d \qquad \text{to} \qquad \mathbf{a}' = \mathsf{location}_{l',d} \ .$$

There is at least one thing wrong with that answer; it is incorrect. The reason is, of course, that it reports the latest sighting recorded in $\mathbf{l}'$ as opposed to the actual location of d in $\mathbf{C}'$. This discrepancy is realistic, as discussed in earlier chapters. In this simple model there is, however, no bound on how wrong an answer can be. In this case, device d just moved up one level so the answer is a reasonable approximation. (Had it moved down, we would have had an even better approximation.) Of course, this may not be the case, d could have moved an arbitrary number of times since the query was generated, so it may be anywhere in the, say, building represented by the state of $\mathbf{C}'$. The second problem with the answer is that location is not of sort $\mathcal{K}_{\mathbf{A}'}$, it is of sort $\mathcal{K}_{\mathbf{L}'}$ only. This can be easily mended, though. We merely add location in the signature of $\mathbf{A}'$; atomic with arity $(0 \to 2)$ just like in $\mathbf{L}'$. Sorting helps us to discover such non-sensical terms (bigraphs), just like traditional static type systems of programming languages.

We continue with step 4 of the scenario: another "whereis" query is issued, this time for device d'. This query will linger in $\mathbf{a}'$ forever as there is no such device in $\mathbf{c}'$ or $\mathbf{l}'$ so the `whereis` rule in $\mathbf{A}'$ can never fire. We may think of changing the rule that generates these queries in $\mathbf{A}'$ to only ask for the location of existing devices. This, however, relies on the unrealistic assumption that a location-aware application always knows which other mobile devices exist, not just in its vicinity, but everywhere.

Now, assume that d' actually exists in $\mathbf{c}'$, for instance in location l. The state $(\mathbf{c}' \parallel \mathbf{p}' \parallel \mathbf{a}')$, where $\mathbf{p}' = \mathbf{l}'$ because $\mathbf{S}'$ has no state of its own, of the system is:

$$\mathsf{loc}_{l'}(\mathsf{loc}_{l}(\mathsf{dev}_{d'}) \mid \mathsf{dev}_{d}) \parallel \mathsf{devs}(\mathsf{location}_{l',d}) \parallel (\mathsf{location}_{l',d} \mid \mathsf{whereis}_{d'}) \,.$$

Step 5 of the scenario; $\mathbf{S}'$ discovers that d' occurs in $\mathbf{c}'$ but not in $\mathbf{l}'$ and reacts by rule `observe_new` to produce:

$$\begin{aligned} & \mathsf{loc}_{l'}(\mathsf{loc}_{l}(\mathsf{dev}_{d'}) \mid \mathsf{dev}_{d}) \\ \parallel\ & \mathsf{devs}(\mathsf{location}_{l',d} \mid \mathsf{location}_{l,d'}) \\ \parallel\ & (\mathsf{location}_{l',d} \mid \mathsf{whereis}_{d'}) \,. \end{aligned}$$

Now, rule `whereis` may fire, let us assume that it does, to produce:

$$\begin{aligned} & \mathsf{loc}_{l'}(\mathsf{loc}_{l}(\mathsf{dev}_{d'}) \mid \mathsf{dev}_{d}) \\ \parallel\ & \mathsf{devs}(\mathsf{location}_{l',d} \mid \mathsf{location}_{l,d'}) \\ \parallel\ & (\mathsf{location}_{l',d} \mid \mathsf{location}_{l,d'}) \,. \end{aligned}$$

And so completes the scenario. Clearly, we may move d down into l, generate another "whereis d" query, update $\mathbf{l'}$ via $\mathbf{S'}$, and finally fire `whereis` in $\mathbf{A'}$ to produce:

$$\begin{aligned} & \mathsf{loc}_{l'}(\mathsf{loc}_{l}(\mathsf{dev}_{d'} \mid \mathsf{dev}_{d})) \\ \| \quad & \mathsf{devs}(\mathsf{location}_{l,d} \mid \mathsf{location}_{l,d'}) \\ \| \quad & (\mathsf{location}_{l',d} \mid \mathsf{location}_{l,d'} \mid \mathsf{location}_{l,d}) \ . \end{aligned}$$

There is a catch, though. We have cheated a little bit in $\mathbf{S'}$. The rule for updates does not adhere to the requirement that $cod(R) = cod(R')$, because the global outer name l' is not present in $cod(R')$. Thus, we must augment R' by tensoring with $^{l'}/$ in the $\mathbf{A'}$-part to obtain:

```
loc_l(dev_d | [0]) || devs(location_l',d | [1])
  ->
loc_l(dev_d | [0]) || devs(location_l,d  | [1]) * l'/
```

where * is the ASCII-representation of tensor product $\otimes$. The same is the case for the $\mathbf{S'}$-rule `lose`.

Now, consider the last state again. Evidently, there are now two answers for the location of d in $\mathbf{a'}$. We would like some sort of garbage collection to get rid of obsolete answers. One idea is to alter the rules in $\mathbf{A'}$ to update answers instead of just adding them, but then we would like to somehow ensure that the answers are not updated before they have been read by the person or program asking $\mathbf{A'}$ for results. This issue is, however, beyond the scope of our example. Another option would be to introduce time stamps on queries and answers, but then we move into the territory of extending Bigraphs, as discussed in Chapter 5.

A final point to make is that we still have to use closed links on the top level of the system to maintain the invariant that an outer name x in one BRS is the same as outer name y in another BRS. Recall that locations and devices are now identified by outer names. Even experienced modellers may not have realised all of these points in advance, even though it is a simple scenario for a small model.

The reason that we have not implemented this model and simulated it using the BPL tool is that there are still some problems with matching of links in the tool. Instead, in the next section, we recast the abstract model to a version not using links, we implement that model, we extend the scenario, run some simulations, and comment a bit on the some of the code.

6.3 Simulation with the BPL tool

The following link-less model exhibits different problems from the previous one, but let us not get ahead of ourselves in the discussion of these issues. Here is the model.

6.3.1 World, C''

The world is represented much like before, with one-step moves up and down the location hierarchy. The difference is that locations and devices are now identified by controls instead of links. To keep track of the identity of a location we place its identifier inside a node with an id control. Identifiers are representations of non-negative integers, as discussed in Chapter 4. Devices have exactly one node inside each of them; an identifier. Thus, there is no need to encapsulate those. In ASCII notation we denote the sort $\mathcal{K}_{\mathbf{C}''}$ of $\mathbf{C}''$ by C, for instance.

```
signature world =
  sig
    loc         : active  (0 -> 0)
    dev         : passive (0 -> 0)
    id          : passive (0 -> 0)
    i0,i1,i2... : atomic  (0 -> 0)
  end

using world

rule moveup  = (* sort: C *)
  loc(id([0]) | [1] | loc(id([2]) | [3] | dev([4])))
    ->
  loc(id([0]) | [1] | loc(id([2]) | [3]) | dev([4]))

rule movedown = (* sort: C *)
  loc(id([0]) | [1] | loc(id([2]) | [3]) | dev([4]))
    ->
  loc(id([0]) | [1] | loc(id([2]) | [3] | dev([4])))
```

The state is of form

$$\mathsf{loc}(\mathsf{id}(n) \mid \mathsf{loc}(\ldots) \mid \mathsf{dev}(\mathsf{id}(m)) \mid \ldots) \,.$$

Do also notice that the control loc representing a location is now active (and has no free ports). This is to allow sufficient matches in nested locations.

This is a subtle point. In the definition of reaction, Definition 4.3 of [JM04], it is stated that the context must be active. That means active everywhere. So no holes may reside under passive controls. If so, the context is not active, and even though there may be a match, there will be no reaction. Nevertheless, Milner and Høgh comment that the theory of Wide Reactive Systems of which Bigraphical Reactive Systems are an instance, just as well supports the case where only the part of the redex to be rewritten must lie at an active site in the context. These more refined rules are not implemented in the BPL tool, though. Therefore, we make our model more liberal by changing the activity of the control loc to 'active'. Below, we will identify the point in the simulation, where this makes a crucial difference.

6.3.2 *Sensor,* $\mathbf{S''}$

The sensor still has three rules and no state. Two of the rules are actually schemas. There is a rule for each choice of $n \in \mathcal{N}$, that is, for each control i0, i1, i2 and so forth. In fact, there are infinitely many such rules. As can be seen in the implementation, cf. App. A.2.1, we have explicitly given the needed instances of some of these rules.

```
signature sensor =
sig end

(* rule schema, one for each n in i0,01,i2... *)
rule observe_update = (* sort: C,L *)
  loc(id([0]) | [1] | dev(n)) ||
  devs(location(l([2]) | d(n)) | [3])
    ->
  loc(id([0]) | [1] | dev(n)) ||
  devs(location(l([0]) | d(n)) | [3])

(* need a NAC to ensure that the iN node in [2]
   is not in [3] *)
rule observe_new = (* sort: C,L *)
  loc(id([0]) | [1] | dev([2])) ||
  devs([3])
    ->
  loc(id([0]) | [1] | dev([2])) ||
  devs([3] | location(l([0]) | d([2])))
```

```
(* need a NAC to ensure that the iN node in [3]
   is not in [0] *)
rule lose = (* sort: C,L *)
  loc([0]) || devs([1] | location(l([2]) | d([3])))
    ->
  loc([0]) || devs([1])
```

Unfortunately, we need negative application conditions (NACs) to control the use of two of these rules, i.e., to ensure that they only fire under the proper conditions. We need this to enforce that reaction can only happen when something, a control, is *not* present somewhere in the system. Such a condition can not be expressed directly in Bigraphs. This is a weakness in comparison with other graph rewriting formalisms; a point which will be discussed in the next chapter on related work, Chapter 7.

The problem arises because we can no longer use the programming trick of declaring a link closed in some part of the system, which implies that no other controls link to that name. The usual way to try to enforce such constraints is to use a sorting. Unfortunately, we see no way to impose such NACs by sorting. The trouble is that we do not wish to outlaw the redeces as bigraphs, but only to disallow the contexts that enable a reaction *under the wrong circumstances*. The hacker's solution to this problem would be to program the simulation in such a way that reactions with these rules are only attempted when the structural condition that the NACs represent is met. For our purposes we settle for a solution where we produce all of the matches – also the illegal ones – and then just pick the legal ones.

We do best to consider invariants of the system. These are structural conditions on the state (bigraph) of the system, which must hold for the initial state of the system, and must also be preserved during reaction. The invariants are:

- If a node has control loc then it contains at least one node, which has control id.
- If a node has control id then it contains exactly one node, which has control iN. (iN ranges over i0, i1, i2...)
- If a node has control dev then it contains exactly one node, which has control iN.
- If a node has control location then it contains exactly two nodes, one which has control l and the other control d.

- If a node has control l then it contains exactly one node, which has control iN.
- If a node has control d then it contains exactly one node, which has control iN.
- If a node with control iN occurs inside a node with control id, then there does *not* exist a node with the same control iN inside a node with control dev, and vice versa.

They should appear quite natural. We will construct the initial state of our assembled Plato-graphical system such that these invariants hold.

6.3.3 *Location model,* L''

The location model is now link-less. Apart from that it is as before.

```
signature repr =
  sig
    devs     : passive (0 -> 0)
    location : passive (0 -> 0)
    l        : passive (0 -> 0)
    d        : passive (0 -> 0)
  end
```

The state is on form:

$$\mathsf{devs}(\mathsf{location}(\mathsf{l}(n), \mathsf{d}(m)) \mid \ldots) \; .$$

Finally, we present the application part.

6.3.4 *Application,* A''

This part now has rules for generating queries, and it is link-less.

```
signature agent =
 sig
   findall     : atomic  (0 -> 0)
   whereis     : passive (0 -> 0)
   i0,i1,i2... : atomic  (0 -> 0)
   id          : passive (0 -> 0)
   location    : passive (0 -> 0)
   l           : passive (0 -> 0)
```

```
   d           : passive (0 -> 0)
 end

using agent

rule findall = (* sort: L,A *)
  devs([0]) || findall
    ->
  devs([0]) || [0]

rule whereis = (* sort: L,A *)
  devs(location(l([0]) | d(id([1])) | [2])) || whereis([1])
    ->
  devs(location(l([0]) | d(id([1])) | [2]))
  || location(l([0]) | d(id([1])))

rule genFindall = (* sort: A *)
  [0] -> [0] | findall

(* rule schema, a rule for each n in i0,01,i2... *)
rule genWhereis = (* sort: A *)
  [0] -> [0] | whereis(n)
```

Again, we use rule schemas to generate q rule for every $n \in \mathcal{N}$.

6.3.5 *Rules preserve invariants*

By inspecting each rule of this model, one can be convinced that the invariants are preserved under reaction.

This completes the abstract, link-less model. We will now present the initial state and the scenario to be simulated. During the presentation we will interleave a few code snippets to illustrate how the simulation proceeds.

6.3.6 *Initial state and extended scenario*

This time around we have provided the full model up front. The model exhibits a few characteristics different from the previous model.

The initial state of the system $\mathbf{c}'' \parallel \mathbf{l}'' \parallel \mathbf{a}''$ is as follows:

$$\begin{aligned} \mathbf{c}'' &= \mathsf{loc(id(i1) \mid loc(id(i2) \mid dev(i3) \mid dev(i4)))} \parallel \\ \mathbf{l}'' &= \mathsf{devs(location(l(i2) \mid d(i3)))} \parallel \\ \mathbf{a}'' &= \mathsf{1}\ . \end{aligned}$$

This initial state respects the invariants stated above. We wish to simulate the following extended scenario:

1. A "whereis" query for device i3 is issued.
2. A move of device i3 occurs in $\mathbf{c}''$.
3. An answer to the query appears in $\mathbf{a}''$.
4. Another "whereis" query is issued, this time for device i4.
5. $\mathbf{S}''$ discovers that device i4 occurs in $\mathbf{c}''$ but not in $\mathbf{l}''$ and reacts.
6. An answer to this query appears in $\mathbf{a}''$.
7. A "findall" query is issued.
8. An answer to this query appears in $\mathbf{a}''$.

As mentioned, the model and the above scenario have been implemented and simulated using the BPL tool. The code can be found in Appendix A.2.1.

Given the initial state the BPL tool provides 17 matches with rule `genWhereis`. Perhaps that is surprising. Inspecting the matches reveals that most of the matches are essentially the same, but for the placement of a single site in the context. Even for link-less bigraphs there are many possibilities. When linking is involved there is an explosion in the number of possibilities. Recall that links can be renamed, even back and forth. We pick a match that suits us to obtain an altered $\mathbf{a}''$:

$$\begin{aligned} \mathbf{c}'' &= \mathsf{loc(id(i1) \mid loc(id(i2) \mid dev(i3) \mid dev(i4)))} \\ \mathbf{l}'' &= \mathsf{devs(location(l(i2) \mid d(i3)))} \\ \mathbf{a}'' &= \mathsf{whereis(i3)}\ . \end{aligned}$$

For the next two reactions, we pick the match that moves device dev(i3) up in $\mathbf{c}''$, and then return the location of device dev(i3), as recorded in $\mathbf{l}''$. The

result is:

$$\begin{aligned}\mathbf{c}'' &= \mathsf{loc(id(i1) \mid loc(id(i2) \mid dev(i4)) \mid dev(i3))}\\ \mathbf{l}'' &= \mathsf{devs(location(l(i2) \mid d(i3)))}\\ \mathbf{a}'' &= \mathsf{location(l(i2) \mid d(i3))}\ .\end{aligned}$$

Like before, the answer is imprecise. Step 4: Where is device dev(i4). Denote the result s_4:

$$\begin{aligned}\mathbf{c}'' &= \mathsf{loc(id(i1) \mid loc(id(i2) \mid dev(i4)) \mid dev(i3))}\\ \mathbf{l}'' &= \mathsf{devs(location(l(i2) \mid d(i3)))}\\ \mathbf{a}'' &= \mathsf{location(l(i2) \mid d(i3)) \mid whereis(i4)}\ .\end{aligned}$$

Finally, we arrive at a crucial point. It is here that we discover the design choice made in the BPL tool. Given refined rules as mentioned above, we would have the following two matches:

$$\begin{aligned}C_1 &= -_0 \parallel -_1 \parallel \mathsf{location(l(i2) \mid d(i3))|whereis(i4)}\\ p_1 &= \mathsf{i1} \parallel \mathsf{loc(id(i2) \mid dev(i4))} \parallel \mathsf{i3} \parallel \mathsf{location(l(i2) \mid d(i3))}\\ C_2 &= \mathsf{loc(id(i1)} \mid -_0 \mid \mathsf{dev(i3))} \parallel -_1\\ p_2 &= \mathsf{i2} \parallel \mathsf{1i4} \parallel \mathsf{location(l(i2) \mid d(i3))} \mid \mathsf{whereis(i4)}\ ,\end{aligned}$$

with the redex $R = \mathsf{loc(id(}-_0\mathsf{)} \mid -_1 \mid \mathsf{dev(}-_2\mathsf{))} \parallel \mathsf{devs(}-_3\mathsf{)}$. It is easy to verify that $s_4 = C_1 \circ R \circ p_1 = C_2 \circ R \circ p_2$, as required.

The BPL tool merely provides the first one of these as the context of the second one is not active (at all sites). When inspecting the matches one sees that in C_2 site $-_0$ is within a node with control loc, and if this is passive then that site is not active. This is not an insight about the model, but about the implementation of the BPL tool. It is interesting because it illustrates the importance of activity for modelling – it is like a programming construct or a primitive in a calculus.

Now, performing the change in $\mathbf{l}''$ with rule `sobsnew` and then completing step 6 with the rule `whereis` we obtain:

$$\begin{aligned}\mathbf{c}'' &= \mathsf{loc(id(i1) \mid loc(id(i2) \mid dev(i4)) \mid dev(i3))}\\ \mathbf{l}'' &= \mathsf{devs(location(l(i2) \mid d(i3)) \mid location(l(i2) \mid d(i4))}\\ \mathbf{a}'' &= \mathsf{location(l(i2) \mid d(i3)) \mid location(l(i2) \mid d(i4))}\ .\end{aligned}$$

Evidently, there are two observations in $\mathbf{l}''$ and two corresponding answers in $\mathbf{a}''$. This is about to change as we generate a "find all" query and answer

it by rules `genFindall` and `findall`, respectively. The resulting state, after steps 7 and 8, where the query is replaced by an answer, is as follows:

$$\begin{aligned}
\mathbf{c}'' &= \mathsf{loc(id(i1) \mid loc(id(i2) \mid dev(i4)) \mid dev(i3))} \\
\mathbf{l}'' &= \mathsf{devs(location(l(i2) \mid d(i3)) \mid location(l(i2) \mid d(i4))} \\
\mathbf{a}'' &= \mathsf{location(l(i2) \mid d(i3)) \mid location(l(i2) \mid d(i4)) \mid} \\
&\quad\ \mathsf{location(l(i2) \mid d(i4)) \mid location(l(i2) \mid d(i3))}\ .
\end{aligned}$$

Behold the mess. It is impossible to tell – by just observing $\mathbf{a}''$ – which answer(s) correspond to which querie(s). We will not go further into this detail here, because the aim of the simulation was to show that the model is an approximation or abstraction of the full model, albeit a coarse one. The techniques we use in this simulation code scale to the full model.

6.3.7 *Properties of the abstract model*

The model is very simple so its properties are not very deep or elaborate. Nevertheless, we may consider whether the answer to the two queries are correct. In the simulation above, they are. In this case it seems that there is no way to get wrong answers, but to be certain we should try out all possible interleavings of the events and consider the resulting matches and states. Surely, picking different matches – for example moving device dev(i4) up instead of device dev(i4) – will make a drastic difference to the rest of the scenario. We would obtain other answers. Not wrong ones, just other ones.

Did the small, controlled simulation tell us anything that we did not know beforehand? Did it unveil something that we did not expect? Unless we had anticipated all the above-mentioned issues beforehand, the answer to the questions would be positive. Then imagine what simulation can help us realise on a grand scale.

6.3.8 *Concluding remarks*

A feature that we did not use in this case study was that of *tactics*. Tactics are a way for the user of the BPL tool to realise priorities of rules. With a tactic one can specify in which order, how often, and under which simple conditions (an if-then-else construct where the Boolean condition depends on the successful application of another rule) a rule should be applied. This is likely a useful programming construct for larger systems.

This concludes our treatment of simulation. The BPL tool helps us realise decompositions that we may otherwise have neglected and it catches mistakes like, for instance, missing idle names in rules and that some

controls should be active. We claim that through our experiments we have justified that bigraphical simulation is useful for exploring the design space of even small systems. We retain a hope that the BPL tool can scale to execute realistic location models. Nevertheless, enhancements of Bigraphs and the BPL tool are needed to reach the goal of having a model for ubiquitous computing. We discuss future work within simulation in Chapter 8.

Now, we take a brief detour considering a case study for the application part **A** of our Plato-graphical model.

6.4 Case study: a tour guide

In our model of Chapter 5 we had a simple application part *A*. The Plato-graphical framework allows for much more elaborate examples than that, though.

In Appendix A.2.2 there is SML code implementing the core functionality of the tour guide GUIDE of [CDMF00]. That is a real-life location-aware system in Lancaster, England. The code is written using only the parts of SML that can be translated into MiniML, as shown on Chapter 4. We have included it to demonstrate that the Plato-graphical idea does indeed scale to realistic systems.

The tour guide is a piece of software running on hand-held computers, which are lent out to visitors to Lancaster. The guide is context-sensitive in that it has knowledge of the physical location of the device on which it is running, and it can be customised according to user preferences. This knowledge is used to display information and perform services specific to both user and location. For example, if a user is interested in particular historic artefacts such as castles, then the GUIDE device is able to construct a walking tour which takes account of this interest. It also supplies directions on how to get from one location to the next. As the user arrives at each destination the device displays a description of the site from a historical perspective. The devices obtain their information and communicate via a wireless communications link. They even support interactive services such as ticket booking or enquiries, communications with other users and with the tourist information service, and access to the Internet.

We could, in principle, insert the implemented tour guide as part *A* of our Plato-graphical model and have our MiniML-to-bigraphs compiler produce a bigraphical version of it. We have not done so yet because the BPL tool currently can not run examples of that magnitude – the MiniML code unfolds to a very large bigraph with thousands of nodes. The location model presented in Chapter 5 is even worse. The reason that it can not be

run is that the matching [BDGM06] of agents with rules is a computationally difficult problem – there are simply too many ways of decomposing bigraphs. We need to improve the efficiency of the BPL tool; the BPL tool was implemented very closely according to the theory and thus designed to be correct. We have, of course, tested the tour guide program in isolation to convince ourselves that it works properly.

6.4.1 Properties

One of the goals of modelling and simulation is to be able to provide guarantees about a model, and ultimately, the system that the model represents. In the case study of the tour guide one might consider which properties are desired to have established, and which are feasible to establish, either formally or by simulation.

Here is a list of possible such properties:

- Obtain information about the attractions there are at the current location. Property: The guide displays the correct information.
- Navigation help: "You are in Half Moon Bay, to get to Humbugs Sweetshop you should...". Property: The guide displays the shortest path.
- From a list of attractions can the guide display a tour that visits all locations. Property: "The shortest route, all attractions visited."
- The interface to the location model part **L** of the system; what must **L** provide to be a reasonable model of **C**. See Leonhardt's dissertation [Leo98] for a discussion on this.
- The relation between **C**, **L** (and **A**). Can one say something about how **C** and **L** develop in relation to each other?

It is future work to actually test whether these properties, and perhaps others, hold for the model. It seems impossible to do so formally by hand, however, simulation may provide us with sufficiently good answers.

In [RCS06], sensors, actuators, an application framework, and environment modelling are claimed to be typical ubiquitous computing components. Hence, our efforts with location models, which have all of these (a sensor part, communication from **A** to **L**, the MiniML framework for applications, and a model of the environment **C**), are not far from ubiquitous computing. We may even encompass the diversity of devices in ubiquitous computing by simply introducing more controls into our model.

7

Related Work

In this chapter we discuss related work for the second part of this dissertation, namely with respect to modelling and simulation.

7.1 Modelling

In the second part we have taken an *experimental* approach to testing whether Bigraphs is useful for modelling and programming context-aware (and in particular location-aware) systems. In this chapter we assess to what degree other formalisms can answer the challenge of modelling and programming context-aware systems.

Bigraph theory has roots in process calculi, and in particular Action Calculi [Mil96] and Reactive Systems (RSs) [Lei01, LM00b], but it is formulated in category theory. Thus, we consider formal approaches to explicit context-awareness based on process calculi and RSs to be closely related work. We also review a logic for specifying context-aware systems, which has a tuple space-based middleware supporting it. Due to the completeness (from theory to practice) of this approach we merit the presentation of it ample space.

We do *not* consider Algebraic Graph Transformation (AGT) [EEPT06] in this part because there has, to the best of our knowledge, not been attempts to formalise context-awareness via AGT. It is, however, related work regarding Bigraphs, so relating Bigraphs and Plato-graphical models to AGT will be relevant, at least from a theoretical point of view, at some point. There are also several toolkits and middlewares available to support the implementation of context-aware systems, and some systems have such support built into them. We refer to [JPR04] for an overview of these as we do not in this piece of research consider implementations. Studying such systems in more detail could become relevant when implementing a context

toolkit on top of a bigraphical rewrite engine. In our work we do not try to develop new location models and thus do not consider work in that area to be related, i.e. relevant for this chapter. We have merely provided a digest of that literature in Chapter 2 of this part.

Having narrowed the scope of what we consider related work in this part, we proceed to give a method for treating the selected works.

7.1.1 Method

We treat each piece of related work in turn. For each piece of related work we take the following approach:

1. Report on the purpose/aim of the work and its strengths and weaknesses. Include in this part the aspects that are important for modelling and programming context-aware and location-aware systems.
2. Evaluate the work with respect to how the formalism can be used to model and program context-aware (location-aware) systems:
   - Negative context information.
   - Control structures.
   - Representing a location topology.
   - Queries on the topology.
   - Interaction between a location-aware application and the location topology.
   - The modelling effort.
3. Reasoning in practise.

The first item is natural. We try to tailor the treatment of each piece of related work such that we can draw on it when relating the work to our own. We have pinpointed six items that are important to consider when estimating a formalism for modelling and programming location-aware systems, and location models in particular. This focus on the location aspect of context reflects the focus of our own work, and we are aware of the fact that some of the related works treated here capture a more general notion of context. The third item is about reasoning which is one of the main reasons for taking a formal approach to the study of context-aware systems. We emphasise that we wish to reason about concrete systems, i.e., really experiment with and challenge the techniques to give guarantees about real systems. Admittedly, this is a hard task.

At a later stage when more applications have been studied (and the formalisms are perhaps more advanced) it would be interesting to try to estimate how well they can be used for simulation, but for now we restrict ourselves to the evaluation above.

We proceed to treat each piece of related work in turn, and then we draw conclusions based on the treatment.

7.1.2 Context calculi versus process calculi

In [RJP04] the difference between context calculi and process calculi is considered to be the presence of constructs to explicitly model context interaction. Another way to express the difference is given in [Bra03], where process calculi are described as formal theories of concurrent, distributed systems taking advantage of algebraic reasoning, and context calculi should separate process behaviour from the (multiple notions of) computational context. This is our criterion for picking related calculi. This rules out traditional process calculi such as the π-calculus.

7.2 Context UNITY

In this section we treat the Context UNITY framework of Roman, Julien, and Payton [RJP04, JPR04].

7.2.1 Report

Context UNITY is a specialisation of Mobile UNITY [RM02, RMP97] to provide constructs to reason about interaction with the context. There are two goals; (1) to simplify development of context-aware applications, and (2) to gain a better understanding of the essential features of the context-aware computing paradigm. Context UNITY programs can be translated into Mobile UNITY programs (with the exception of non-deterministic assignment), which means that Context UNITY can largely be considered a syntactically sugared version of Mobile UNITY. Underlying the Mobile UNITY syntax is a translation into a first-order Hoare-style temporal logic [RM02].

The next two sections (7.2.1 and 7.2.1) give a quite detailed walk-through of Mobile UNITY and the additions to obtain Context UNITY. We have devoted ample space for this treatment, but the reader may be satisfied with the résumé given first.

Context UNITY, very briefly

"Context UNITY represents an application as a community of interacting agents." [RJP04]. Each agent is uniquely identified and its behaviour exclusively defined by a program describing its interaction with variables. An agent interacts with its context by reading and writing (actuation) special variables and can itself decide which parts of the context that it finds interesting. The variables are governed by agent-specified (guarded) rules, thus separating the management of an agent's context from its internal behaviour. The unpredictability of context-aware systems is implemented by non-deterministic assignment statements. Agents have a location implemented by a special variable, which can be manipulated by the agent itself (subjective movement) or by the operational environment (objective movement). Agents run concurrently. A simple museum guide system was briefly sketched in [RJP04].

Mobile UNITY

We review Mobile UNITY as a basis for Context UNITY highlighting some important features to give the reader an intuition of the approach. Mobile UNITY captures the notion of location and movement across logical spaces while providing formal reasoning via assertions. To this end the formalism has a notation for expressing mobile computations and a logic for reasoning about their temporal properties. Thus, Mobile UNITY can be said to extend the UNITY [Cha88] model of concurrent computation by adding constructs for component location and transient interactions among components. Mobile UNITY aims to decouple a program's internal functionality from its interaction with the computational context.

Here is an overview, which will be explained in more depth afterward, of a Mobile UNITY program:

- A *system* consists of *program declarations*, a *components* section, and an *interactions* section.
- A program has a *location*, declares *local variables* with their initial values, and specifies clauses to control subsequent assignment to these variables. A clause can be the asynchronous conditional *when* or the synchronous conditional *reacts-to*. Assignments can also be enclosed in an *inhibit-when* statement, which disallows assignment under certain conditions.
- The components section instantiates the programs.

- The interactions section defines how the programs can interact because it has access to the variables of the programs.

In Mobile UNITY one specifies a *system* consisting of *programs* (processes) running in parallel, non-deterministically scheduled in a weakly-fair manner. The key elements of program specification are variables and (labelled) conditional multiple assignment statements.

Programs are sets of conditional assignment statements, and each program has a location, which is a variable outside the Mobile UNITYmodel (and thus parametrises the model). Programs have *declare* and *initially* sections much like imperative programs. The *assign* section defines a program's behaviour. The transient interactions in Mobile UNITY consist of four additions to UNITY: *Transactions* (sequential critical regions), globally unique *labels* (on statements), *inhibitors* (strengthening of guards by predicates), and *reactive statements* which are to be executed to fixed-point, interleaved, after any other executed statement including those in transactions. One observes that the reactively augmented statements make up the basic atomic state transitions of the Mobile UNITY model. The *components* section defines which programs that make up the system and their initial locations. The *interactions* section defines interaction between the programs. Programs interact solely via shared variables.

Regarding location it is worth remarking that programs have subjective (or local) movement, i.e., they can access their own location variable. This can, e.g., be used to enforce co-location during process communication.

Shared variables are transient in the sense that sharing is controlled by a predicate (*when*) guarding the assignment.

Comparing reactive statements and when-conditions, one one can think of reactive statements as providing context information in an eager manner, while when-conditions are lazy. The logic is used for reasoning, e.g. about safety and liveness properties of the system, see [RM02] for a formal definition of these properties.

As mentioned in [RM02], the Mobile UNITY notation is very expressive, but should be restricted in practice to obtain, e.g., termination guarantees and a more efficient implementation.

From Mobile UNITY to Context UNITY

In [RJP04] it is stated that context models should have the following properties:

- Expansive: The scope of context of a particular agent should not have *a priori* limits.
- Specific: It must be possible to specify tailor-made context definitions for an agent – context definitions should be modifiable.
- Explicit: Agents control their contexts – this requires an explicit notion of context.
- Separable: An agent's context specification can be separated from its behaviour specification.
- Transparent: The agent should be freed from the operational details of discovering its own context.

In Context UNITY context is defined from the perspective of a single component. One can think of a component (agent) as a state transition system; state change (representing change in context) occurs spontaneously without agent control (cf. the weakly-fair scheduling). Behaviour (and state) of a program is defined exclusively through its interaction with variables.

There are three important variable types. *Internal*: These are not accessible/visible outside the agent. *Exposed*: Public variables that can be part of other agents' context. (These have access control associated with them.) *Context*: Variables that directly model, access, and modify context in a program by getting and putting information from/to exposed variables of other agents according to the agent-specific *context rules* (see below). In addition to context variables a program now has a *context section*, which provides rules to manage an agent's interaction with its desired context, i.e., sensing of information from the operational environment[1], and affecting other agents by impacting their exposed variables. The context section thus secures decoupling of an agent's internal behaviour and management of its context; it is said to allow *projections*, i.e., local change can imply global change.

An agent can also feed back information into its own context. We take a short digression into exposed variables before continuing with nondeterministic assignment. Exposed variables have an in-built access control policy; exposed variables consist of six components, one being a function from the reference agent's[2] *credentials* (see below) to a set of operations on that variable (e.g. read or write). Four such exposed variables are built-in:

[1] In Context UNITY, the operational environment consists of everything that could potentially effect a program (computational agent), and the part that does is named context.

[2] By reference agent we mean the agent whose context we are considering [JR06].

Location, type, agent identifier, and credentials. The type variable holds the program's name, while the agent identifier is the unique global ID of the agent. In the credentials variable a profile of attributes for the program is stored for querying by other agents regarding access to their exposed variables.

Returning to non-deterministic assignment; to handle the lack of *a priori* knowledge about the operational environment, Context UNITY introduces non-deterministic assignment statements to be used in context rules. A context rule governs the interactions associated with a certain context variable. The rule can quantify over variables that are place-holders for other agents' exposed variables (via the *uses* construction), and it can define restrictions (via the *given* construction) and interactions (via the *where-becomes* construction) with these. Context rules can be declared *reactive*.

Systems in Context UNITY also have a *governance* section, which contains rules to capture behaviours that have universal impact across the system. This happens through assignment to exposed variables, and governance rules are assumed to be safe (and thus do not involve credentials). Objective (so-called global) movement can, e.g., be implemented via governance rules.

Formal differences

Formally, there is only one difference between Mobile UNITY and Context UNITY, namely an additional proof rule for non-deterministic assignment to context variables modelling the unpredictable real world. It is claimed that the other constructs can be translated into Mobile UNITY constructs [RJP04]. This sounds plausible.

Middleware support

Two middlewares have been used in conjunction with Context UNITY; EgoSpaces and LIME.

EgoSpaces [JR02, JR06] is a middleware that provides context information to applications in an abstract form. EgoSpaces evolved from LIME. As the authors write in [JR06]: "LIME requires strong assumptions about the operating environment that fail to hold as the number of devices, connections, and the degree of mobility grows." EgoSpaces is claimed to overcome these weaknesses. EgoSpaces is compared to the Context Toolkit in [JR06], and there EgoSpaces is found to be more suitable for supporting development of context-aware applications (in an *ad hoc* network scenario) because

it addresses an application's need to dynamically discover and operate over a constantly changing context.

An important notion in EgoSpaces is a *view*. A view is a projection of all data available to the reference agent. Views can be created, redefined, and deleted, and are defined over network, host, agent, and data constraints. Views are updated when agents access them, and thus provide asymmetric coordination. EgoSpaces provides triggered reactions and also "migrate", "duplicate", and "event" primitives, since these have been found common and useful in practise. EgoSpaces is implemented in Java using tuple spaces (via Elights), the CONSUL framework is used to collect context information from sensors and other agents close by, and the SICC protocol for making a network tree and sending messages. There is, surprisingly, no discussion of whether such a location tree suffices for these purposes, we would suspect not. Performance simulation has been done via the OMNet++ discrete event simulator.

The LIME [MPR06] coordination middleware is an implementation of a LINDA-like tuple space calculus supporting mobility, and it has been specified using Mobile UNITY, and also in CRSs (see below). To demonstrate how LIME can be used as a context-aware middleware supporting context-aware applications, an example application for tracking users via GPS called TULING [MP04] was modelled. A lightweight version of LIME is Limone [FcRH04], which "...centers the coordination tasks around *acquaintances*, and knowledge of specific coordinating partners is essential to Limone's functionality." [JR06]. Thus, it fails to capture the unpredictability of *ad hoc* networks.

Numerous other middleware systems exist, but we refer to [MP04] for an overview of these, since middleware as such is not the focus of this dissertation. We do, however, mention Klaim, which is a formalism (process calculus) to support computing with mobile processes and explicit localities. It uses LINDA-like tuple spaces. Furthermore, a modal logic has been developed. There is also a programming language X-Klaim, based on Klaim, for programming distributed applications with mobile code. A compiler from X-Klaim into Java using the Klava package (run-time system for Klaim) exists.

The Context UNITY logic

As mentioned, the logic underlying Context UNITY is a first-order Hoare-style modal logic. Properties are proven by constructing proof trees.

7.2.2 *Evaluation*

Negative context information is represented by using inhibitors, i.e., guards on context variables. Thus, rules only fire when the guards allow it.

There are different types of variables and guarded assignments to these. We also have transactions, inhibitors, context rules, credentials and so forth. Context UNITY is convenient for programming.

All agents are "on the same level" topologically, like in CRSs (see below). Thus, it is not clear how one would represent a hierarchical location topology.

If we implement a graph-based topology, then symbolic range queries are not supported.

Probably, agents would interact with the topology via context variables. An agent could then inform other agents of its movements.

The modelling effort may not be low because of the location topology problem, but Context UNITY is well-suited for many other context aspects: Context interaction (sensing and actuation) is sharply separated from internal program behaviour, and is possible via context variables (in the context section). Non-deterministic assignment statements capture the *a priori* unpredictable operational environment.

To the best of our knowledge no location model has been formalised in Context UNITY, but we imagine that formally it would resemble the approach of [Leo98], because a first-order logic is used there also. However, some examples using location information have been encoded.

7.2.3 *Reasoning in practise*

To the best of our knowledge, the logic has not been used to reason about the examples implemented. All Context UNITY constructs can be translated into logic formulae, and it would then in principle be possible to build proof trees establishing certain properties. It does seem cumbersome, and formally handling and proving things about systems with non-determinism is notoriously hard.

This concludes our discussion of Context UNITY.

7.3 Contextual Reactive Systems (CRSs)

In this section we treat Braione and Picco's theory of CRSs [Bra03, BP04].

7.3.1 Report

CRSs is a generalisation of Leifer and Milner's RSs [LM00a] in the sense that interactions between computational agents (processes) and the context are disciplined. In CRSs it is possible to specify the computational contexts under which a class of behaviours is allowed, and the contexts under which it is not. CRSs is a category theoretical approach inspired by that of Reactive Systems and Bigraphs [LM00a, JM04]. The motivation for CRSs is two-fold:

- To separate the process behaviours from the computational context.
- To allow the specifier to define the notion of context and the rules governing how it affects the processes.

The goal of CRSs is to devise a formalism for modelling real middleware, in particular LIME. It is claimed that process calculi have been successfully used for specifying the semantics of coordination models and languages, but that these do not sufficiently address the modelling of a changing computational context; many such calculi have a rather rigid built-in context notion, i.e., a tight coupling between context and process. We agree.

In CRSs computational steps are described as transitions that rewrite terms and thus change the state of the system. The trouble with RSs, in the scenario of context-awareness, is that every computational context may host any interaction which makes it difficult to represent dynamic, subjective behaviour. We think of RSs as specifying the *internal* semantics of a system. In CRSs, it is possible to specify the computational context under which a class of interactions is allowed (via *enablers*), and under which it is not (via *inhibitors*). Context is a first-class element of the formalism. We recite the relevant definitions (1, 2, 3, and 4) of [BP04] to make the following discussion more clear.

Definition 7.1 (Reactive System)**.** *A* reactive system *(RS) is a* $(\mathcal{C}, I, \mathbf{R}, \mathcal{D})$ *quadruple, where* $\mathcal{C}$ *is a category,* $I \in Ob\mathcal{C}, \mathbf{R} \subseteq \cup_{x \in Ob\mathcal{C}} \mathcal{C}(I, x) \times \mathcal{C}(I, x)$*, and* $\mathcal{D} \leq \mathcal{C}$ *is composition-reflecting, i.e.,* $D_0 D_1 \in Mo\mathcal{D} \implies D_i \in Mo\mathcal{D}$*,* $i = 0, 1$*.*

The morphisms of $\mathcal{C}$ are contexts ("terms with a hole"), and *ground* contexts are processes ("terms where the hole is replaced by a compatible sub-term", by morphism composition), denoted by $\mathcal{C}(I, x)$. Notice that contexts have exactly one hole. $\mathcal{D}$ specifies *reactive contexts*, i.e., contexts under which a rule may fire, and $\mathbf{R}$ is the set of *elementary* rewrite rules on processes; rewrite rules are pairs (l, r) of ground contexts, where l is named *redex* and r *contractum*. Extending (composing) elementary rules

with reactive contexts, along with the following reaction relationship, yields *composite* rules:

Definition 7.2 (Reaction relationship)**.** *The* reaction relationship, $\rightarrow$, *is defined as follows:* $a \rightarrow a' \iff \exists (l, r) \in \mathbf{R},\ D \in Mo\mathcal{D}\ .\ a = Dl\ \wedge\ a' = Dr$.

This relationship contains both elementary and composite rules of the RS so $\mathbf{R} \subseteq \rightarrow \subseteq \cup_{x \in Ob\mathcal{C}} C(I, x) \times C(I, x)$. A motivating example of context-aware printing is given and it is remarked that "it is not possible to forbid the reduction of a redex based on the properties of the context it is immersed in" [BP04]. (It is this example that we encoded in Chapter 3.) To alleviate this, CRSs are proposed which allow elementary rules to be extended only be some instead of any active contexts. The following definition captures exactly this:

Definition 7.3 (Contextual Reactive System)**.** *A* contextual reactive system *(CRS) is a* $(\mathcal{C}, I, \mathbf{R}, \mathcal{D}, \mathcal{D}[\![l, r]\!])$ *quintuple, such that* $(\mathcal{C}, I, \mathbf{R}, \mathcal{D})$ *is a RS, and* $\mathcal{D}[\![l, r]\!]$ *is a function mapping any elementary rule* $(l, r) \in \mathbf{R}$ *to a composition-reflecting sub-category of* $\mathcal{D}$.

The reaction relationship is altered correspondingly:

Definition 7.4 (Reaction relationship for Contextual Reactive Systems)**.** *The* reaction relationship, $\rightarrow$, *is defined as follows:*
$a \rightarrow a' \iff \exists (l, r) \in \mathbf{R},\ D \in Mo\mathcal{D}[\![l, r]\!]\ .\ a = Dl\ \wedge\ a' = Dr$.

We see that different elementary rules may have different contextual constraints so some expressivity was gained. We remark that internal transitions performed by a group of processes still does *not* affect the surrounding context, i.e., there is no *actuation*.

Using *enablers* and *inhibitors*, for (elementary) rules, one can control what may and may not be present in the context for a rule to fire. Essentially, an enabler is a set, closed under morphism composition, which has as elements predicates on tuples. Such a set precedes a rule and guards applications of this, so to speak. Likewise for inhibitors. We refer to [Bra03, BP04] for the formal definitions, as we here merely wish to pass on intuition. We do, however, recite an example from [BP04]. Here is a rule printing on a 'raw' printer, which does not fire if a Postscript printer is present, assuming a **print** primitive and a simple tuple space process calculus (cf. Table 1 of [BP04]):

$\{- \mid \langle v \rangle\ .\ v \neq \text{pr:ps}\}^\star \quad \mathbf{print}(\text{txt}).P \mid \langle \text{pr:raw} \rangle \longrightarrow P \mid \langle \text{job,txt,raw} \rangle \mid \langle\ \text{pr:raw} \rangle$

where '−' denotes a *hole* in a context, and angle brackets denote tuples; $\langle v \rangle$ is a tuple with value v, for example. This rule can be rewritten using an inhibitor:

$$\{- \mid \langle \text{pr:ps} \rangle\}^{c\star} \quad \textbf{print}(\text{txt}).P \mid \langle \text{pr:raw} \rangle \longrightarrow P \mid \langle \text{job,txt,raw} \rangle \mid\langle \text{ pr:raw} \rangle$$

The small c signifies that the tuples mentioned in the set must *not* be present in the context if the rule is to fire. Further, an enabler can be specified:

$$\{- \mid \langle \text{pr:ps} \rangle\}^{c\star} \quad \{- \mid \langle \text{pr:raw} \rangle\}^{i} \quad \textbf{print}(\text{txt}).P \longrightarrow P \mid \langle \text{job,txt,raw} \rangle$$

It looks simple, but is underpinned by a few technical constructions. What has happened in these two reformulation steps is essentially to move information from the term to the context of a rule.

Application of CRSs

CRSs have been used to formalise the core of the LIME middleware [Bra03], and also as a tuple space process calculus based on Linda. The idea is to represent context by a global tuple space, thus separating specification of behaviour from specification of context where it may occur. In [MP04] there is a description of the TULING application which shows how location context can be made available in LIME. Thus, for location context information, LIME can be used much like a context toolkit, i.e., facilitate programming of mobile applications that need access to location information.

7.3.2 Evaluation

Negative context information can be represented by inhibitors.

CRSs can be thought of as a meta-calculus, like Bigraphs. This has the advantage that different domain-specific calculi can be encoded. There are no in-built control structures to facilitate convenient programming. One can achieve control by representing calculi with control structures, but then the control structures depend on the calculus that is formulated as a CRS. As an example: For a process calculus based on Linda there will be operations for interacting with tuple spaces, and these will provide some control of computation.

Only flat location topologies can be represented — if represented as a term — and it is not feasible to have a hierarchical space reflected in the syntactic term structure in the general case, as mentioned in [BP04]. This

means that whichever calculus we wish to represent as a CRS we can only have flat terms.

Considering this limitation we can not support range queries in a natural way, nor nearest neighbour queries. Position and navigation queries do seem feasible, but we lack structure and this is likely to render programming of queries harder than in Bigraphs and thus not convenient. As stated in [MP04], a query such as "find all components within a radius r from point (x, y)" can not be performed because that would require a range search inside the tuple space, and LIME only provides value matching [MP04].

How interaction between a location-aware application a and the location topology t would be realised depends on the calculus represented as a CRS. No matter the choice of calculus, terms will be flat and a will live in the same system and at the same level as t.

The modelling effort would probably be high because we lack structure for our specific purpose despite encoding a suitable calculus.

7.3.3 Reasoning in practise

It is unknown to what extend the techniques of [LM00a] can be used in the setting of CRSs, i.e., whether operational congruences can be derived automatically (from a RS to a LTS, i.e., from internal to external semantics). In [Bra03] it was necessary to restrict the definition of bisimulation to ensure that bisimilar processes have sufficient contexts in common. Furthermore, assessing whether adding negative context information increases expressiveness, needs to be explored. No logics exist for CRSs.

Collecting these facts it is fair to say that reasoning in practise is not feasible yet, and no such attempts have been made to the best of our knowledge.

7.4 A calculus for context-awareness (CAC)

In this section we treat Zimmer's calculus for context-awareness (CAC) [Zim05].

7.4.1 Report

CAC "is a process calculus, whose aim is to describe dynamic systems composed of agents able to move and react differently depending on their location." [Zim05]. CAC features a hierarchical term structure like Mobile

Ambients [CG00], and a generic *multi-agent synchronisation* mechanism inspired by the Distributed Join calculus [FGL+96], we remark that locations are organised in a tree. One can think of the calculus as a hybrid between the two calculi just mentioned, except that it also features non-local process synchronisation. The motivation is to develop a calculus that models how devices interact in a uniform way in wireless networks or mobile ad hoc networks.

Ambients are called *agents* and these represent locations, which are either physical or logical units of computation. Agents have *definitions* that enable enclosed processes to perform reductions. The main feature of CAC is multi-agent synchronisation of tuples of values on *named channels*. Agents are not directly aware of their environment, but inform it of their capabilities by asynchronously sending *atoms*.

The environment has definitions consisting of *rules* (think join patterns) to perform global synchronisation on these captured atoms; this is novel with respect to the Distributed Join calculus, where synchronisation happens locally and not across agent boundaries. There is a notion of priority (or scope) of patterns, namely that the deepest rule (pattern) matches first.

Agents can move by the *go* primitive, but must give the explicit path to the destination because movement happens one step (up or down in the tree) at a time.

There is no way to open or close agents, just like in Boxed Ambients [BCC01]. Definitions of a particular agent are activated (by a reaction rule) by adding them to the enclosing agent's definitions.

Reaction rules rewrite an agent if an enclosed process matches one of the agent definitions (under some particular restrictions). We remark that contexts can have any number of holes, and that parallel composition in CAC is not commutative, which according to [Bra03] is unusual, with which we agree. The reason for this is that the names in the redex of a rewriting rule are bound point-wise in the process expressions in the reactum of the rule.

Example encodings and expressiveness

A small location-dependent printing example is given to illustrate how a process interacts with its enclosing agent. Further, a form of "remote procedure call" and also a small packet routing protocol are encoded via continuations.

Expressiveness is investigated by encoding a monadic asynchronous π-calculus with replicated input, and a λ-calculus.

7.4.2 Evaluation

A basic model has been developed where agents may use different notions of computation on different physical locations.

In the current version of CAC it is not possible to express negative context information. The author suggests to add negated terms in pattern rules. This would be like adding "not-controls" (or co-controls) to Bigraphs, which is inelegant because then there would have to always be one or the other. Introducing a notion of inhibitor like in CRSs or a sorting like in Bigraphs is preferable.

The control structures in CAC are somewhat like those in Bigraphs; a tree hierarchy of terms (agent processes), and a way to link agents – namely by channels and name restriction. Rules (patterns) are used to control reactions. These rules are part of the agents so contexts are not really separated from processes, at least not to the same extend as in CRSs or Bigraphs. Anyhow, when programming with the calculus these structures and the movement primitive are useful. We emphasise the fact that each agent has its own set of rules, which can grow as other agents move into it so that it can activate their definitions. This is the way a changing context is modelled.

The location topology is a tree just like in Mobile Ambients. In Bigraphs there is a forest of trees available.

Queries by an agent — for example a location-aware application — happen on the structure in which it resides. Probably, one would want to program an auxiliary process to traverse the tree and collect information. To do this the agent must know the topology of the entire tree because the *go* primitive needs an explicit path. This is inconvenient and does not harmonise with the unpredictability of context in general, but it is reasonable for a location hierarchy like a building. Apart from that, programming in CAC does resemble programming natively in Bigraphs, and is thus likely to be equally inconvenient. In principle, it should be possible to support three of the four query classes discussed in Chapter 2, but the nearest neighbour queries require some notion of distance. Interaction is between processes in the tree structure.

The modelling effort regarding the hierarchy is low because there is only one tree, but it is high with respect to queries. We find that encodings of larger examples are required to further test the modelling capabilities of the calculus.

7.4.3 Reasoning in practise

More work is needed with respect to the behavioural and equational theory of CAC. Thus, the calculus is not yet ripe for reasoning about systems in practise.

7.5 A formal model for context-awareness (CONAWA)

In this section we treat Baun Kjærgaard and Bunde-Pedersen's effort to define a formal model for context-awareness [KBP06a, KBP06b] based on Mobile Ambients.

7.5.1 Report

Like other approaches, [KBP06a, KBP06b] argue that we lack formal support for realistic context-awareness. Furthermore, it is claimed that existing calculi [BP04, RJP04] only deal with very limited notions of context, and that a flat space structure does not suffice while being difficult to navigate. Also, context is not just physical location, but also logical information.

The approach taken in CONAWA takes origin in the Ambient calculus, but instead of having one tree representing space it has several so-called *views*, much like the place graphs of bigraphs. The intention is to have one tree (view) for each category of context information needed by the application, e.g., locations and printer types.

In CONAWA, ambients are divided into two syntactic classes: *Context* and *reference* ambients.

Context ambients (views) have unique names, are static in the sense that they can not move (navigate views), and can only be created (declared initially) and opened (a standard capability).

Reference ambients are embedded in context ambients and navigate these by exercising the capabilities *in*, *out*, *enter*, *exit*, *coenter*, *coexit*, and *open*. *in* and *out* are not observable by the context, whereas *enter* and *exit* are. Using the co-capabilities requires the context to allow these movements.

Capabilities are instrumented with two pieces of information; a Boolean expression over contexts to define which contexts the owner of the capability can be performed with respect to, and the names of the reference ambients the owner of the capability can exercise the capability on. A wildcard name is included to match all reference ambients.

Reference ambients can be replicated and can input/output names locally. A reference ambient will have a single (for consistency) presence in one ore more views simultaneously by a reference, e.g., a printer ambient

has a type and a location. The capabilities *in* and *out* of the Ambient calculus are proposed extended to enable navigation in several views at once. A reference ambient navigates views by explicitly giving the path to collect context information. This reflects the idea that computation is seen as embedded in a number of contexts at the same time.

Communication is local, i.e., an ambient (or rather a reference to an ambient) may communicate with other ambients which are its siblings or parent in the tree where it currently resides. They communicate through the *ether* of each view — much like a tuple space — with no channels involved. Actions and capabilities are restricted by Boolean expressions over contexts.

Name scoping and general output paths have been left out of the calculus for simplicity. This model is parametrised over the notion of "proximity".

7.5.2 *Evaluation*

The authors evaluate their calculus by modelling examples of the four types of context-aware applications described in [SAW94], to which we return in Chapter 8. For now, it is enough to know that the four application types are categorised from a user interface (UI) perspective according to whether they provide information or supply commands, and whether they are invoked manually or as a reaction to the current context. The authors find that representatives (in the domain of "pervasive health-care") of all four types could be modelled in CONAWA. Two thematic examples are "find the nearest available doctor" and "update a context/view using a reference process".

Contexts are special uniquely named ambients. Reference ambients may move around in the contexts by exercising capabilities. These capabilities may be instrumented with a "Boolean" expression stating which contexts they match and which they do not, i.e., in which contexts the capability can be used. This is decided by naming the matching contexts, and putting a negation sign in front of the names of non-matching contexts. This is not the same type of negative information as for example the inhibitors of CRSs. The difference is that inhibitors limit reaction to certain contexts where something is not present, but "Boolean" expressions discriminate named contexts and not their contents. This is a gain of introducing views. We also have views in Bigraphs in that a place graph is a forest of trees that can be made uniquely identified by requiring each tree to have a unique control on the root node.

There are some important control structures; views, (guarded) capabilities, and agent references. Agent references allow the specifier to refer to an agent to partake in different contexts, i.e., different views on the world/situation, simultaneously. The guards can be used to control in which views a given capability can be exercised, which resembles programming with conditionals.

The location topology is a forest of uniquely identified trees, i.e., a collection of views.

The only query that lacks support is "nearest neighbour". You have to consider the entire system to formulate single ambients. Specifically, an agent has to explicitly give a path for moving, which may be an unreasonable assumption because it requires detailed a priori knowledge of the whole operational environment and not just the context at hand. This does not reflect the *ad hoc* nature of the real world, but is reasonable enough for a location hierarchy.

Interaction between a location-aware application a and the location topology t can happen by programming a to traverse t. An idea is to have an auxiliary agent collect this information. The authors suggest to introduce designated reference agents to update contexts, i.e., to act as carriers of sensor information. An example is given in the discussion of [KBP06a], which requires the ability for reference agents to output capabilities, and not just names. Having reference agents invoke these "sensor agents" could be considered actuation. Still, there is no *representation* of the world (like **C** in Plato-graphical systems).

We remark that reference ambients can not remove themselves from views, so a device will always have some location once it has been located once. This is not a problem for location models, as seen in Chapter 5.

The modelling effort is probably on level with that of CAC since both are Ambient-based process calculi, albeit with some differences (patterns versus guarded capabilities and views).

7.5.3 Reasoning in practise

No behavioural or equational theory has been established for the calculus, nor any expressivity results. There is no formal semantics of the calculus, merely a few examples of what reduction rules could look like.

We conclude that much work is needed before any formal reasoning can be carried out.

Here are some suggestions for corrections:

- Considering Table 1 of [KBP06a], presenting the syntax of CONAWA, it can be seen that there is no base case for the syntactic categories **C** and **R**, thus the inductive definition is not well-founded.
- The important example in Figure 11 is not syntactically correct because (1) a reference agent 'FNDAP2' is (illegally, see Table 1) used as prefix to a capability, and/or (2) the square brackets do not match.

It should, however, be possible to correct these errors.

7.6 Other approaches

We mention a work where a location model is formalised in first-order logic, two works in progress, and one piece of research formalising context from the viewpoint of artificial intelligence (AI).

- A formal location model in Z/Eves [Leo98].
- The Agent Distributed π-calculus (AgDpi) [Hen05, Hen04].
- The N$^\sharp$ programming language effort [WBB06].
- "Formalizing Context" [MB97].

In appendix C of [Leo98] three location services are specified in the Z formalism, which is a first-order logic [MS97]. Here, we merely give the reader a taste of the approach. A service consists of a location hierarchy, updates on this hierarchy, and some queries. We briefly consider the symbolic one. First, two object types are declared; LOCATION and OBJECT. A location hierarchy is then declared as an asymmetric and transitive inclusion ordering. Predicates in locations corresponding to the relevant spatial relationships are also defined. Here is a parametrised location query which for all located-objects at a given location:

$$target? : LOCATION \land result! : \mathcal{P}(OBJECT) \implies$$
$$result! = \{x : OBJECT \mid (x, target?) \in locatedAt\}$$

where 'locatedAt' is a predicate which decides whether a given object is in a given location, and $\mathcal{P}$ is the power set. A sighting operation is also defined, along with some other queries. Thus, set theory is used as a programming language.

AgDpi is Distributed π-calculus (Dpi) with *nominal agents*. In AgDpi there is mobile code running inside nominal agents. Dependent types

are used to enforce selective access (read/write capabilities) to resources. Locations are unique and organised in a flat structure. Communication is local and authenticated (via types). A special kind of channel *disc* is used by agents to discover local resources, and then the agents act accordingly. This work is worth following should it progress from the current "work in progress" status, e.g., emerge as a full-fledged process calculus.

In [WBB06] a first step is taken toward a programming language for pervasive applications based on the Ambient calculus. The language is called N$^\sharp$ because its syntax resembles C$^\sharp$ or Java. Communication is between ambients, and processes are asynchronous. Named ports are adopted from the π-calculus to facilitate easy message passing. A prototype compiler exists.

In [MB97] context is formalised as a first-class object, and can be thought of as a generalisation of a collection of assumptions (in a Gentzen style logic). A context may even correspond to an infinite and only partially known collection of assumptions. The point of origin is artificial intelligence, and the formalism used is a first-order logic. There is no clear relation between this work and our field of research so we refrain from further discussion of this work.

7.7 Concluding remarks

We structure our remarks according to the method above.

7.7.1 Evaluations

Negative context information, e.g., in the form of inhibitors, is certainly a useful feature. Whether it proves necessary for modelling ubiquitous systems in Bigraphs and Plato-graphical models is uncertain. We suggest more modelling experience for deciding this.

Control structures decide how convenient the programming task is. In the calculi where hardly any are available, it seems unrealistic to model and program realistic systems.

Hierarchical location topologies should feature in calculi for location-awareness. Considering how other aspects of context may very well be hierarchical in nature, for example the organisation of a company, we conjecture that flat topologies are not sufficient.

Like for location, queries on the context are best supported if the context is structured. Furthermore, control structures help.

Interaction between a location-aware application and the context topology can become complicated if the application itself is part of the topology. Separating concerns, as in Plato-graphical models, is useful.

The modelling effort is high when programming directly in meta-calculi. One can gain control structures by encoding other more domain-specific calculi, though. As far as we know, it is a novelty to explicitly represent the world as a system in its own right, as done in Plato-graphical models. It is this feature that is the basis of our simulation idea.

7.7.2 Reasoning in practise

None of the works considered here have been used for reasoning in practise. Nor have Bigraphs or Plato-graphical models. Tool support seems to be required to really make progress in this area.

7.7.3 Summa summarum

On a high level we can say that further experimentation with large examples is needed, and that tool support is essential in this effort. Much work in improving the theories and tools also persists.

7.8 Simulation

In [RCS06], initial work on the design of a generic simulation tool for ubiquitous computing is described. Like in our work, they focus on sensors, actuators, an application framework, and environment modelling. They divide their architecture into three levels; an application level, an interface which provides APIs similar to those of real ubiquitous systems (for example a location system), and a simulator. Ubiquitous applications interact via an API with the interface, which in turn interacts with the simulator. Hence, the application programmer can have his program simulated, and then install that program on a mobile device without any alteration. This is very neat, but the system is still under development and in need for experimentation. Furthermore, it is not underpinned by theory.

This concludes our review of related work.

8

Future Work in Modelling and Simulation

This chapter is divided into future work on modelling and simulation.

8.1 Modelling

We have identified some directions for future work within modelling.

- Characterise context-awareness in terms of Plato-graphical models and enrich our model to support this.
- Model a real-life system.
- Create a more extensive list of properties one wishes (to prove or guarantee) for context-aware systems.
- Improve the BPL tool; efficiency and sorting.
- Investigate formal reasoning about Plato-graphical systems, perhaps by studying a form of bisimulation between BRSs.
- Prove the dynamic correspondence between Ξ programs and their bigraphical images under $[\![\cdot]\!]_X$.
- Enhance Bigraphs as a theory.

We discuss each one in turn.

8.1.1 Characterising context-awareness

Motivated by the fact that the notion of context is still ill-defined [DA00], we strive for a finer taxonomy of context with the purpose of a "context checklist" for applications, which could help to define needed components

in a library for context-aware programming. We believe that characterising the context types of [SAW94] in terms of Plato-graphical models will aid in understanding context-awareness as such by sharpening the definitions. By formalising an application's context interaction as interaction between Plato-graphical components we gain precision.

We briefly recall the four types of context-aware applications that are mentioned in [SAW94]. The types are along two axes: Manual vs. automatic, and information vs. command. Manual and automatic refer to whether the user has to do something to make the application either fetch information or perform an action specified by the command.

Proximate selection has to do with finding or emphasising the located-objects that are nearby, and is a manual information task.

Automatic contextual reconfiguration is an automatic information task that adds or removes components (typically software) or alters connections (typically wireless) depending on the context.

Contextual information and commands are commands whose execution depend on the context – printing to the nearest printer will have a different result depending on the user's location.

Context-triggered actions are simple 'if-then' rules used to specify how context-aware systems should adapt, and context-triggered actions are invoked automatically according to these rules. This is enough knowledge for our purposes.

Now, consider the following setup (suggested by Niss) depicted in Figure 8.1.

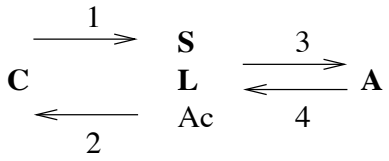


Figure 8.1: The four categories of context-aware applications as Plato-graphical interaction.

We imagine the following four interactions:

1. The proxy **P** possesses a sensor **S**, which senses reconfigurations in the context **C** and informs the model **L**.

2. **P** is extended with an *actuator component* (Ac), which can affect **C** on behalf of **A**, i.e., make it reconfigure.

3. The agent **A** is informed of relevant context change by **P** (**L**).
4. **A** affects **L**, i.e., makes it change its conception of the context information.

One can think of (2) as actuation, e.g., if the agent wished to turn on the light in a dark room. (4) represents the ability to override the model if it, e.g., has an inaccurate or wrong conception of the context. (3) can be either "manual" or "automatic" (to use the terms of [SAW94]), with the manual case being the agent asking for information, and the automatic case being some sort of event or call-back. (1) can likewise be divided into manual and automatic.

We claim that this setup generalises [SAW94]. "Automatic contextual reconfigurations" are handled entirely in **C**. "Context-triggered actions" is the call-back of (3) mentioned above. "Proximate selection and contextual information" is the manual version of (3). "Contextual commands" are more complicated. We think of this as a sequence of interactions; first **A** asks **P** for the relevant context information and then it issues a command by (2) and (4) above.

Once established the characterisation should be challenged by capturing other informal characterisations of context-aware interactions. In [Sch95] the following questions for determining *situations* are emphasised: Where the user is, who the user is with, and which resources are nearby. The components are device agents (that maintain status and capabilities of devices), user agents (that maintain user preferences), and active maps (that maintain location information of devices and users). Another work to draw challenges from is [DA00] where the computing environment consists of CPUs, devices, and network connectivity. There is also a notion of user environment characterised by location and nearby people. Furthermore, the physical environment such as lighting and noise level is important. The model requirements here include interpretation, acquisition, and storage (history) of context.

This characterisation may serve as a framework for comparing concrete context-aware models (of realistic systems).

8.1.2 Modelling real-life systems

We could try to establish a collaboration with engineers working on a real-life ubiquitous system. This way, we would encounter massive challenges to drive forward our research, and perhaps be able to solve some real problems for real people.

Another idea is to model a protocol for delivering messages in MANETs such as Geocast [DR03]. Modelling a protocol like Geocast seems to require devices to contain messages, and some sort of reachability information inherent in the topology. Collaboration with people in model checking and verification of wireless networks seems like the way to go here.

8.1.3 A list of properties

We should create a list of properties we want (to prove/guarantee) for context-aware systems. All properties should be relevant for real-life systems and preferably also be provable by the reasoning principles available. Some properties may not be provable with the current techniques so they may give rise to research of new reasoning principles for Bigraphs or Plato-graphical models.

8.1.4 Tool support

In the BPL group at the ITU, some members are currently working on a prototype implementation of BRSs. Normalisation and matching of Binding Bigraphs has been implemented [BDGM06]. Perhaps it is an idea to implement an add-on for Local Bigraphs.

This tool will be crucial for simulation purposes because realistic examples easily become too large to handle manually, let alone reason about. We need tool support to truly conduct experiments with real systems. We have already defined a translation from Ξ_{sugar} into the implementation of binding bigraph terms. Extending the BPL simulator with fully automatic location event generation – perhaps inspired by the generic location event simulator of [SC02].

8.1.5 Formal reasoning

Proving properties about the bigraphical location model could be desirable, but it is unclear which properties we wish to prove and with which techniques. Certainly, one possible direction is to make precise some criteria for when components of a Plato-graphical system can be substituted (while maintaining properties of the system as a whole). This has to do with bisimilarities between BRSs.

We may wish to prove properties such as, e.g., access control. This may involve using BiLog [CMS05], or perhaps access control could be ensured via sorting (only allowing devices with a particular access token to enter

a room with a matching token), and then proving that the system can not place an illegal device inside a protected room, by rule induction. Further studies in desirable properties of context-aware systems are needed. We have uniqueness of device locality by invariant (rule induction).

A technique for securing certain properties of our models could be to use sortings further. We could perhaps impose a building sorting to ensure that certain locations (perhaps identified by a internal type control) are not within certain other locations, e.g., we do not wish for buildings to be within rooms. To combine sortings, Debois, intuitively, combines predicate sortings via conjunction by a pullback construction (Proposition 4 of [BDH06]), so we can combine an additional sorting with the Plato-graphical sorting.

8.1.6 Dynamic correspondence

Ξ-programs and their images under $[\![\cdot]\!]_X$ evaluate in one-to-many correspondence, i.e., the bigraphical representation takes one or more steps for each Ξ-reduction step. Thus, we would like to prove *something like* the following conjecture.

Conjecture 8.1.
$\forall e, \sigma, X, s, g.\ (\exists\, e', \sigma'.\ \langle e, \sigma\rangle \to \langle e', \sigma'\rangle \ \wedge\ [\![\langle e', \sigma'\rangle]\!]_X = /\vec{Y}.g \mid s) \implies$
$([\![\langle e, \sigma\rangle]\!]_X \rightarrowtriangle^+ /\vec{Y}.g \mid s)$, *where* $\mathsf{fv}(e) \cup \mathsf{fv}(e') \subseteq Y \subseteq X$, *and* $\rightarrowtriangle^+$ *is the transitive closure of* $\rightarrowtriangle$.

The idea of a proof should be: Analyse each of the cases of possible reaction in Ξ. We need a substitution lemma for the cases where evaluation results in a substitution. If we can prove such a lemma, then the result can be lifted to evaluation contexts by Lemmas 8.2 and 8.3.

Lemma 8.2. *For any evaluation context E, term e, and set X such that* $\mathsf{fv}(E[e]) \subseteq X$, *it holds that* $[\![E[e]]\!]_X = [\![E]\!]_X \circ [\![e]\!]_X$.

Proof. A proof should be by structural induction on E.

Lemma 8.3. *This lemma should correspond to Definition 4.8.*

A proof should be by structural induction on evaluation contexts. One could also consider studying other programming language issues in the setting of Bigraphs. It would probably be wise to do it in as simple a setting as possible. We do, however, not see this as important for our current endeavours.

8.1.7 Enhancing Bigraphs

Currently, the following extensions are on the wish list:

- DAGs: Replace the place graph (a forest of trees) with a DAG. The motivation was given in Chapter 5, where organising a building with rooms, wings etc. was a little troublesome.
- Time: Timed automata may inspire, we think of [D'A99].
- Continuous space (hybrid systems): Possible works of inspiration (apart from the ones mentioned just above): [AD94, ACHH93, Hen96, DB96].

We discuss each item in turn in a little more detail after considering the overall purpose of them.

The visions mentioned in Chapter 3 remain; DAGs, time, continuous space, and probabilistic information. Enriching the theory of Bigraphs with these aspects is a demanding task, but certainly interesting.

DAGs

We decided to use the place graph for representing the location hierarchy instead of merely by linking. On first thought, it might seem reasonable to hierarchically order floors, wings, rooms, and devices in a building. Choosing one ordering has its drawbacks, however. Should wings or floors be higher in the tree? If we choose floors over wings then we could end up representing each wing on every floor, thereby introducing redundancy. This can, however, be remedied by using DAGs instead of trees. Furthermore, DAGs naturally support "shared locations" as, e.g., an auditorium residing on the floors simultaneously. If we shy away from altering the theory of Bigraphs then DAGs could be implemented using several trees (roughly one for each "location sharing"), but navigating and keeping consistent several such *views* would complicate the modelling effort. In Chapter 7 we discussed a piece of related work, where a sort of pointer is proposed to address this idea.

Time

A notion of time in the model is required to be able to order events, i.e., for $\mathbf{S_X}$ to be able to inform $\mathbf{L_X}$ of the order of sightings to facilitate a closer correspondence of the states $\mathbf{c}_X$ and $\mathbf{l}_X$.

Continuous space

Continuous space has to do with geometric coordinates. It should be possible to determine the geometric whereabouts of located-objects, and to compute metric distances.

Probabilistic information

Stochastic Bigraphs [KMT08] may be a big help in this endeavour.

8.2 Simulation

We would like to test whether the properties of the tour guide mentioned in Chapter 6 hold for the full model of Chapter 5. This will require extensive simulations. This, in turn, will require us to develop further the BPL tool. There are many directions in which we could go:

- Improve the efficiency of the BPL tool by reimplementing the matching engine to *compositionally* build matches from "submatches", i.e., reuse parts of matches in a dynamic programming style.
- Tighten the normal forms for bigraphs to limit the search space.
- Improve the user interface, i.e., provide a more extensive high-level graphical interface than is currently available at the BPL Web page[1].
- Implement support for (semi-)automatically selecting appropriate bigraph matches from a given specification of their desired qualities. This is important when many matches exist and the bigraph is large. One such quality could be simplicity (number of nodes) of the parameters.
- To establish correctness we could link with work on model checking and/or static analysis. The idea would be to write a model checker for (some version of) the bigraphical term language (BTL), or to write a compiler from BTL to the source language of known model checking and static analysis tools.

This concludes the chapter on future work.

[1] `https://tiger.itu.dk:8080/bplweb/`

9

Summary of Modelling and Simulation

In this part, the second part, we have investigated the modelling and simulation of context-aware system in Bigraphs. In particular, we have modelled an extensive location model in Bigraphs by use of Plato-graphical models. Then, we have abstracted that model to obtain a more tangible model for simulation purposes. Finally, we have presented some simulations of the abstract model to illustrate the feasibility of the approach.

On the way, we have developed Plato-graphical models and en encoding of MiniML into Bigraphs. Both theoretical advancements have been motivated by modelling efforts.

Finally, we have given a thorough account of related work and pointed out directions for future work.

In the next part, the third part, we will change focus. The third part is more theoretical in nature than the second part, as we explore type systems (and to some extent sortings) for Bigraphs.

Part III

Type Systems

"There are many ways of trying to understand programs. People often rely too much on one way, which is called 'debugging' and consists of running a partly-understood program to see if it does what you expected. Another way, which ML advocates, is to install some means of understanding in the very programs themselves."

– Robin Milner, Foreword to The Little MLer [FF97].

"If I had learned to type, I never would have made brigadier general."

– Brigadier General Elizabeth P. Hoi

10

Type Systems for Bigraphs

This chapter consists of a paper [EHS09] published at the 4th International Symposium on Trustworthy Global Computing (TGC'08). I produced the results of this paper along with their presentation under guidance of Davide Sangiorgi and Thomas Hildebrandt. I made a major contribution both in the research and writing phase, which is evidenced by a co-author statement accompanying this dissertation. The paper has been insignificantly altered to match the layout of this dissertation, and a few auxiliary definitions have been admitted. Moreover, the full proofs from the accompanying technical report [EHS08] have been included. Finally, the section on future work has evolved.

Abstract

We propose a novel and uniform approach to type systems for (process) calculi, which roughly pushes the challenge of designing type systems and proving properties about them to the meta-model of *Bigraphs*. Concretely, we propose to define type systems for the term language for Bigraphs, which is based on a fixed set of *elementary bigraphs* and *operators* on these. An essential elementary bigraph is an *ion*, to which a *control* can be attached modelling its kind (its ordered number of channels and whether it is a guard), e.g., an input prefix of π-calculus. A model of a calculus is then a set of *controls* and a set of *reaction rules*, collectively a *bigraphical reactive system* (BRS). Possible advantages of developing bigraphical type systems include: a deeper understanding of a type system itself and its properties; transfer of the type systems to the concrete family of calculi that the BRS models; and the possibility of modularly adapting the type systems to extensions of the BRS (with new controls). As proof of concept we present a model of a π-calculus, develop an i/o-type system with subtyping on this model, prove crucial properties (including subject reduction) for this type system, and transfer these properties to the (typed) π-calculus.

10.1 Introduction

Type systems for calculi are important as they can: detect programming errors statically; and classify terms enabling extraction of information that is useful for reasoning rigorously about the behaviour and properties of programs, among other things. Type systems are usually engineered to enjoy subject reduction. The problem is that changing even small details of such a type system might ruin properties. Therefore, to feel confident that a tweak of the type system does not ruin any properties one really has to redo the proofs. This is often tedious. Many such type systems can be considered to be rather *ad hoc* so one would like a uniform way of proving properties of a whole family of calculi, simultaneously.

In this paper we experiment with a novel approach to type systems for (process) calculi, which roughly consists in pushing the problem of designing type systems and proving properties about them (such as subject reduction) to the more abstract level of *Bigraphs* [JM04, JM03] by Milner and co-workers, a meta-model for (process) calculi. The main advantages are:

a meta-model can describe several concrete calculi, therefore one can hope that a result for a meta-model can be transferred to all of these calculi; and understanding type systems at the level of meta-models can help to achieve a deeper understanding of the type systems themselves. The theory of Bigraphs is rich as its expressiveness has been demonstrated in several works in the literature; Petri nets [Mil04b, LM06], π-calculus [Jen07, JM04, JM03], CCS [Mil06a], Mobile Ambients [Jen07], Homer [BH06], and λ-calculus [Mil07]. Importantly for our work, a sound and complete term language exists for Bigraphs [Mil05a, DB06].formulation of

One models a calculus in bigraphs by encoding its terms as bigraphs and representing its *reduction* semantics by bigraphical *reaction rules*. All bigraphs are obtained by combining *elementary bigraphs* via the *operators* of categorical tensor product and composition. An essential elementary bigraph is an *ion*, to which a *control* can be attached modelling its kind (its ordered number of channels and whether it is a guard), e.g., an input prefix of π-calculus. The semantics of a concrete calculus is represented as reaction rules over a *signature* of controls.

A major effort so far has consisted in using Bigraphs to automatically derive labelled transition semantics and congruential bisimilarities for concrete calculi with semantics defined by a reduction relation. In this paper we propose a novel use of Bigraphs – to derive type systems for the concrete calculi. Our approach can be described in three phases: 1) Define a core BRS that can model the family of concrete calculi one is interested in. 2) Develop *bigraphical type systems* (BTSs) for this core BRS and prove their properties (such as subject reduction). 3) Transfer the type systems and their properties onto the concrete calculi of interest. Transferring the type system rules onto a concrete calculus C follows almost directly from the encoding of C's terms into the BRS and from the typing rules of the BTS. Our approach requires a result of operational correspondence between a concrete calculus and its bigraphical model, which is the most basic and fundamental property to have when mapping a calculus into Bigraphs. Hence, we provide a point of origin for studying type systems *for* (not *in*) bigraphs.

As proof of concept we study a *strict* (no summation), *finite* (no replication) and synchronous π-calculus, dubbed sfπ, along with an i/o-type system with subtyping for its bigraphical model. sfπ with i/o-types is well-suited for three reasons: The relationship between sfπ and Bigraphs has been well studied in the literature [Jen07] allowing us to focus on type systems for Bigraphs; sfπ is simple but important because it maintains the essence of message-passing process calculi, and the i/o-type system with subtyping is technically interesting without being very complex. This cons-

titutes a first study of non-trivial types for bigraphs.

Related work In [BDH08], Debois and collaborators define a *sorting* as a functor from a *sorted s-category*, where *sorts* (think types) are assigned to interfaces (objects) as an extra component, into an unsorted s-category. A sorting refines which bigraphs (morphisms) may be composed and thus guarantees a certain structure of the well-sorted bigraphs. Hence, a sorting reduces the set of terms that are considered for reaction. Sortings are *not* defined inductively over bigraphs and give rise to different guarantees than traditional type systems in that they do not attempt to approximate dynamic behaviour of the terms. Thus, it is unclear whether one can recover existing type systems by sortings (in the general case).

In [BS06] Bundgaard and Sassone develop polyadic π-calculus with capability types and subtyping in bigraphs by: defining and proving safe a *link sorting* — called 'subsorting' — which is crucial in securing the desired i/o- and subtyping discipline; extending the theory of Bigraphs by introducing controls on edges to retain the type information of restrictions. They inductively map *type derivations* of form $\Gamma \vdash P : \diamond$ to *sorted* bigraphs by sending processes P to morphisms and typings Γ to sorted objects J. They also derive an LTS yielding a coinductive characterisation of a behavioural congruence for the calculus. A large effort in that work went into the sorting and the derivation of the LTS.

In [IK04] Igarashi and Kobayashi propose a generic type system (GTS) for π-calculus enjoying subject reduction and type soundness. They express typings Γ as (abstract) CCS-like processes and then check the properties on Γ. The GTS is parametrised over a subtyping preorder stating when two types have the same behaviour. By adding rules to the basic subtyping relation a type system instance for deadlock-freedom, among others, is obtained. This approach differs from ours in that they consider type systems for π-calculus and not for a meta-model, but we too wish to transfer general results to a family of calculi. In [KÖ5] König aims at generalising the concept of type systems to graph rewriting and in particular the concepts of type safety, subject reduction and compositionality. By working at the more abstract level of graphs rather than terms the author claims to be able to simplify the design of type systems, however we believe, at the cost of making it more difficult to transfer back and understand the type systems in terms of the concrete calculi.

In our approach we define type systems inductively on bigraph terms and can thus hope to: *directly* recover existing type systems; and have a computer verify whether a typed bigraph term is well-typed or not.

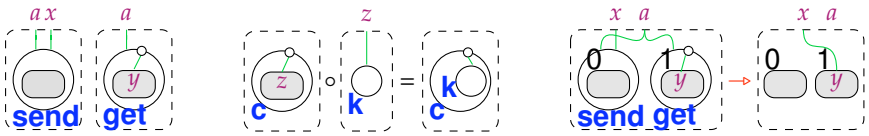


Figure 10.1: The ions send and get, bigraph composition, and the sfπ reaction rule.

Contributions Our main contribution is conceptual: this work is a first attempt in the novel direction of *using bigraphs as a meta-model for type systems* through the first study of non-trivial inductive types for Bigraphs. There are two main technical contributions: an i/o-type system (Table 10.2) for a core BRS capturing the essence of message-passing calculi; and a proof of Subject Reduction (Theorem 10.14) for this type system.

Outline In Section 10.2 we explain the necessary parts of Bigraph theory by example and then we present a model of sfπ. On this foundation we develop an i/o-type system for the model, prove important properties of it, and transfer these to i/o-typed sfπ, all in the main Section 10.3. Finally, in Section 10.4, conclusions are drawn and directions for future work outlined. The full proofs reside in Section 10.A.

10.2 Bigraphs

Bigraphs is a model of computation that emphasis both *locality* and *connectivity* aiming at trustworthy (safe and reliable) computation in global ubiquitous computers [Wei93, BDE+06], in which highly dynamic topologies and heterogeneous devices are prominent. Mobile locality is captured by a *place graph* and mobile connectivity by a link *link graph*, two largely orthogonal structures that combine into a bigraph. The place graph is an ordered forest of trees representing nested locations of computational nodes, and the link graph is a hypergraph representing interconnection of these nodes. Dynamics are added to bigraphs by defining (parametric) reaction rules. Consider Figure 10.1. It depicts two *ions*, bigraph composition, and a reaction rule involving the ions. The two ions, depicted with solid circles, model output prefix and input prefix of π-calculus, respectively. Each ion consists of a *node* assigned a *control* determining its kind. In this case, both controls have two ordered *ports* to which *links* (channels) can be attached. send has its (*free*) ports linked to *local outer names a* (the 'channel' port) and x (the 'datum' port), respectively. *Global names* are like unrestricted names

in π-calculus, whereas local (think abstracted) names reside at *regions/roots* (dotted rectangles) or *sites* (greyed rectangles). get has a *binding port*, which binds a *local inner name* y with lexical scope below this node in the place graph and thus resembles a variable of programming languages. Both ions are contexts with a site (hole) into which another suitable bigraph can be "plugged", yielding another bigraph. This is known as vertical *composition*, $b_1 \circ b_0$, and proceeds by plugging the roots of b_0 into the sites of b_1 (in order), and fusing together the outer names of b_0 with the inner names of b_1, removing the names in the process. The sites and inner names of a bigraph b are collectively called the *inner face* or *domain* ($dom(b)$); similarly, the regions and outer names are called the *outer face* or *codomain* ($cod(b)$). Then, $b_1 \circ b_0$ requires $cod(b_0) = dom(b_1)$. The second column of Figure 10.1 shows an example; given $b_1 = \mathsf{c}_{(z)} : \langle 1, (\{z\}), \{z\}\rangle \to \langle 1, (\emptyset), \emptyset\rangle$ and $b_0 = (\{z\})\mathsf{K}_z : \epsilon \to \langle 1, (\{z\}), \{z\}\rangle$ then $b_1 \circ b_0 : \epsilon \to \langle 1, (\emptyset), \emptyset\rangle$. The interface (or *face*) components are: a *width*; a vector of local name sets drawn from the global name set; and a global name set. They are projected by functor *width*, function *loc*, and function *glob*for , respectively. Function *glob* projects all names of a face.

A notion that is not shown in Figure 10.1 is an *edge* (think restricted name); inner names X and ports P can point to edges E instead of outer names Y, via the so-called *link* map, $link : X \uplus P \to E \uplus Y$. If the name x is *closed* (restricted) then it becomes invisible to the context and any ports which were pointing to this outer name will now instead point to an edge. Edges have no name associated with them, just in the term language to denote which points map to which edges. An edge is a "floating" binder in that it has no lexical scope.

When representing a calculus in Bigraphs one is usually interested in bigraph *terms* that are *ground* and *prime* (also known as *agents*), i.e., bigraph terms that have no sites, no inner names, and outer width 1. Regions (or sites) can be juxtaposed (composed horizontally) by the binary operator *tensor product* $\otimes$, if the operands have disjoint name sets (both outer and inner). A derived operator is the *prime product* $|$, which takes two regions as operands, but allows them to share outer names, and also collapses the two regions into one, while acting as tensor on sites. The third (basic) operation on binding bigraphs is *abstraction* $(X)P$ on a *prime* P, which localises a subset of the global names of P. A face of width 0 without names is denoted by the unique object ϵ.

The reaction rule models communication in sfπ. The *redex* has one region signifying that a send and a get must be collocated and connected to be able to communicate. The *reactum* shows that the bigraph has performed an action, which has depleted the input/output capability. The outer name

a is *idle* in the reactum, i.e., has no preimage under the link map, and the inner name y points to the outer name x, explicitly representing meta-level name substitution in π-calculi.

In Definition 10.1 reaction rules are defined formally. Here, we give some intuition, but the formal definitions can be found in Appendix A.1. It uses the notion of *support equivalence* (see Definition A.20), which for our purposes can be thought of as bigraph equality (see Definition A.7). Intuitively, a context D is *active* w.r.t a (ground) bigraph r if the sites of D into which r is plugged are active, and sites are active if the path to the root in the place graph only has (nodes with) active controls. An *instantiation* essentially maps sites of the redex to sites of the reactum, and also maps the possibly renamed local names of the reactum sites back to the redex sites (see Definition A.29). A *discrete parameter* d is a ground bigraph with no edges and a bijective link map.

Definition 10.1 (reaction rules for bigraphs, [JM04])**.** *A* ground (reaction) rule *is a pair* (r, r')*, where* r *and* r' *are ground rules with the same outer face. Given a set of ground rules, the* reaction relation $\rightarrowtriangle$ *over agents is the least, closed under support equivalence* ($\doteq$)*, such that* $D \circ r \rightarrowtriangle D \circ r'$ *for each active* D *and each ground rule* (r, r')*.*

A parametric (reaction) rule *has a* redex R *and a* reactum R'*, and takes the form*

$$(R : I \to J, R' : I' \to J, \varrho)$$

where the inner faces I *and* I' *are local with widths* m *and* m'*. The third component* $\varrho :: I \to I'$ *is an instantiation. For every* X *and discrete* $d : X \otimes I$ *the parametric rule generates the ground reaction rule*

$$(\ (\mathsf{id}_X \otimes R) \circ d,\ (\mathsf{id}_X \otimes R') \circ \varrho(d)\ )\ .$$

Note that $d = d_0 \otimes \cdots \otimes d_{m-1}$ *with each* d_i *prime. So the instance* $\varrho(d)$ *has factor* $e_j \doteq \varrho_j \circ d_{\bar{\varrho}(j)}$ *for each* $j \in m'$*; it takes the form*

$$\varrho(d) = X \parallel e_0 \parallel \cdots \parallel e_{m'-1} : X \otimes I'\ .$$

Reaction is defined over *concrete* bigraphs, i.e., bigraphs where the nodes and edges have identity. However, we are interested in *abstract* bigraphs. Whenever $b_0 \doteq b_1$ concretely we have $b_0 = b_1$ abstractly. Notice that inner faces are local.

A signature is a set of controls each with: an *arity map* from its number f of free ports to its number b of binding ports; and an *activity map* determining whether it is *active* (an evaluation context), *passive* (guard), or *atomic* (a term).

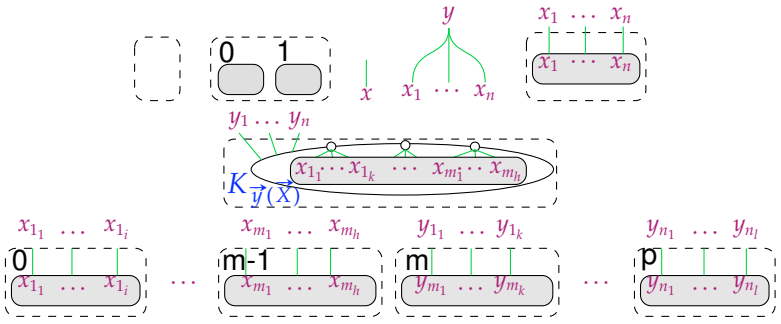


Figure 10.2: The seven elementary binding bigraphs, graphically.

Bigraphs have an algebraic representation. All bigraphs can be generated from seven *elementary* bigraphs combined by (categorical) tensor product and composition. One can think of these elementary bigraphs and the operations on them as basic building blocks (language concepts) for processes and operators on processes. The faces of the bigraphs determine when tensor product, composition, and abstraction are well-defined. Bigraphs are bigraph terms up to the structural congruence given by the axiomatisation. The elementary bigraphs are depicted graphically in Figure 10.2 and as syntactic terms with algebraic faces in Table 10.1.

Notation 10.2 (Placing, linking, wiring, sets)**.** *For interfaces we often omit: names from* placings *(node-free place graphs); widths from* linkings *(node-free link graphs); the enclosing* $\langle$ *and* $\rangle$ *when the width is zero. A* wiring *is a bigraph with zero width generated by composition and tensor of linkings. Curly brackets are often omitted for singleton sets and names on ions. Sets (usually of names or types) are denoted by capital letters such as* X, Y, Z *and* S, T, U*, ranged over by minuscule letters. We write* XY *for the disjoint union* $\uplus$ *of sets* X *and* Y*.*

Definition 10.3 (Flattening)**.** *Given a vector* $\vec{x}$ *of* distinct *names we write* $\{\vec{x}\}$ *for the corresponding (one-to-one) set. Given a vector* $\vec{X} = (X_1, \ldots, X_n)$ *of disjoint name sets we define their disjoint union as* $\{\vec{X}\} \stackrel{\text{def}}{=} \biguplus_{i=1}^{n} X_i$ *.*

Consider Figure 10.2. First row: A barren root 1 is an empty region. When plugging a bigraph with two regions and no outer names into *join*, the two regions are merged into one. Name closure $/x$ acts as a non-lexical binder; it is put on top of a bigraph with a global inner name x and closes (restricts) this name rendering it invisible to the context. *Substitution* y/X links a set of global inner names X to a single global outer name y by a

$\mathsf{1}$	:	$0 \to 1$	barren root
join	:	$2 \to 1$	join two sites
$/x$	:	$x \to \emptyset$	close global outer name x
y/x	:	$X \to y$	link glob. name set X to glob. name y
$\ulcorner X \urcorner$	:	$(X) \to \langle X \rangle$	globalise local outer name set X
$\mathsf{K}_{\vec{y}(\vec{X})}$	:	$(\{\vec{X}\}) \to \langle\{\vec{y}\}\rangle$	ion: local name sets $\vec{X}$, glob. names $\vec{y}$
$\gamma_{m,n,(\vec{X},\vec{Y})}$	:	$\langle m+n, \vec{X}\vec{Y}, \{\vec{X}\} \uplus \{\vec{Y}\}\rangle \to \langle m+n, \vec{Y}\vec{X}, \{\vec{X}\} \uplus \{\vec{Y}\}\rangle$	transpose m with n regions or sites

Table 10.1: The seven elementary binding bigraphs as terms.

hyperlink; y is "substituted for" any $x \in X$; the widths are zero; X is "bound inwards"; and y "binds outwards". A special case is $y/\emptyset$ which introduces an idle name. The *concretion* $\ulcorner X \urcorner$ bigraph globalises a set of local names X, dually to the abstraction operator. Second row: An ion $\mathsf{K}_{\vec{y}(\vec{X})}$ is a prime bigraph with a single node of control K with free ports linked severally to a vector $\vec{y}$ of distinct outer names, and each binding port linked to all local inner names in name set X_i a vector $\vec{X}$ of sets of distinct names. Ions are the essence of BRSs as they usually model the interesting entities of systems or calculi. Third row: A *transposition* $\gamma_{m,n,(\vec{X},\vec{Y})}$ transposes regions keeping their sites and local names. either be put on From here on we think of bigraphs represented as terms.

10.2.1 A bigraphical model of sfπ

We consider sfπ. Following [Jen07] we add an axiom to the usual structural congruence: $\nu x\,(\pi.P) \equiv \pi.\nu x\,P$, if $x \notin (\mathsf{fn}(\pi) \cup \mathsf{bn}(\pi))$. This axiom naturally complements the similar axiom for parallel composition and secures that structural congruence coincides with graph isomorphism yielding a nice graphical representation of bigraphs. Equivalences on processes remain unchanged even though more processes are related by $\equiv$ with this axiom. This axiom is not important for our development. However, we remark that to represent, e.g., a replicated input prefix in Bigraphs one needs an *outward*-binding control [Jen07]. Processes that are α-convertible are identified.

We model sfπ with the BRS of [Jen07] (a signature $\Sigma_{\mathrm{sf}\pi}$ and a set of reaction rules $\mathcal{R}_{\mathrm{sf}\pi}$) but name it $\acute{}\mathrm{B}_{\mathrm{BG}}\mathrm{sf}\pi \stackrel{\mathrm{def}}{=} \acute{}\mathrm{B}_{\mathrm{BG}}(\Sigma_{\mathrm{sf}\pi}, \mathcal{R}_{\mathrm{sf}\pi})$. Prefixes (input and output) are modelled by the *passive* controls send and get of Figure

10.1, because prefixes are guards. get has a binding port. The semantics is modelled by a single reaction rule, where prime product models parallel composition, name closure models restriction, and an *insertion operator* $\lhd$ inserts a wiring into a bigraph b making composition $b \circ b'$ well-defined, typically by "wiring through" (and then localising) outer names of b' that are not to be lexically bound by b. (See Definition A.31 of Appendix A.1.)

Definition 10.4 ($\acute{}\mathrm{B}\mathrm{BG}_{\mathrm{sf}\pi}$, [Jen07])**.**

$$\Sigma_{\mathrm{sf}\pi} \stackrel{\text{def}}{=} \{\, \mathsf{send} : 0 \to 2 \ \text{(passive)}\,,\ \mathsf{get} : 1 \to 1 \ \text{(passive)}\,\}$$

$$\mathcal{R}_{\mathrm{sf}\pi} \stackrel{\text{def}}{=} (R, R', \varrho) = \Big(\mathsf{send}_{ax} \mid \mathsf{get}_{a(y)} : \langle 2, (\emptyset, \{y\}), \emptyset\rangle \to \langle 1, (\{a, x\}), \emptyset\rangle,\ (\mathsf{id}_1 \mid \mathsf{id}_1 \ \lhd\ {}^{x}\!/_{y}) \ \lhd\ a : \langle 2, (\emptyset, \{y\}), \emptyset\rangle \to \langle 1, (\{a, x\}), \emptyset\rangle,\ \mathsf{id}_2\Big)\,.$$

The instantiation $\varrho = \mathsf{id}_2$ has underlying function $\overline{\varrho} : 2 \to 2$ with bijective local substitutions $\varrho_0 : (\emptyset) \to (\emptyset)$ and $\varrho_1 : (\{y\}) \to (\{y\})$. This (parametric) reaction rule is *linear*, because its instantiation is bijective, so no parameters are replicated or discarded. It is parametric to model arbitrary subterms under prefixes. In bigraph *terms* the names a, x, and y are not meta-variables so we stipulate that $a \neq x \neq y$, because the bigraph corresponding to the term is different in the cases where some of these are equal. If a and x need to be identified for a reaction, then the context does it. We name the morphisms (bigraphs) of $\acute{}\mathrm{B}\mathrm{BG}_{\mathrm{sf}\pi}$ *process bigraphs*.

Processes are mapped to bigraph terms by the compositional, semantic function $[\![\cdot]\!]$ of Definition 10.5. For technical reasons the inactive process is modelled by (X), an empty ground local prime.

Definition 10.5 (Encoding sfπ in $\acute{}\mathrm{B}\mathrm{BG}_{\mathrm{sf}\pi}$, [Jen07])**.** *The function* $[\![\cdot]\!]_{(X)}$ *maps every process* P *of* sfπ *with* $\mathsf{fn}(P) \subseteq X$ *into the homset* $(\epsilon, (X))$ *of* $\acute{}\mathrm{B}\mathrm{BG}_{\mathrm{sf}\pi}$ *as follows:*

$$[\![\overline{a}x.P]\!]_{(X)} = \mathsf{send}_{ax} \ \lhd\ \mathsf{id}_X \circ [\![P]\!]_{(X)} \quad [\![P \mid Q]\!]_{(X)} = [\![P]\!]_{(X)} \mid [\![Q]\!]_{(X)} \quad [\![\mathbf{0}]\!]_{(X)} = (X)$$

$$[\![a(y).P]\!]_{(X)} = \mathsf{get}_{a(y)} \ \lhd\ \mathsf{id}_X \circ [\![P]\!]_{(Xy)} \quad [\![\nu\, xP]\!]_{(X)} = /(x) \ \lhd\ \mathsf{id}_X \circ [\![P]\!]_{(Xx)}\,.$$

The translation of P is indexed by a (local) name set $X \supseteq \mathsf{fn}(P)$ that is needed to secure dynamic correspondence between sfπ and the model. This is because reduction in sfπ can discard a channel (name) after use, i.e., reduce the set of free names, but outer faces (of agents) are preserved by bigraphical reaction rules so here the name persists, although idle. (For an example see [JM04].) P will have an image for each choice of X, i.e., countably many bigraphs. Not unusually, the translation requires that

$\mathsf{bn}(P) \cap \mathsf{fn}(P) = \emptyset$ and unique binding names. The model enjoys structural and dynamic correspondence theorems, here combined.

Theorem 10.6 (Correspondence, [Jen07])**.**

1. *The function* $[\![\cdot]\!]_{(X)}$ *is surjective onto the homset* $(\epsilon, (X))$ *of* $\acute{}\mathrm{B}\mathrm{BG}_{\mathrm{sf}\pi}$ *;*
2. $P \equiv Q$ iff $[\![P]\!]_{(X)} = [\![Q]\!]_{(X)}$.
3. *Given* $X \supseteq \mathsf{fn}(P)$*, then* $P \longrightarrow P'$ iff $[\![P]\!]_{(X)} \rightarrowtriangle [\![P']\!]_{(X)}$.

We are now ready to define a type system on $\acute{}\mathrm{B}\mathrm{BG}_{\mathrm{sf}\pi}$.

10.3 A bigraphical i/o-type system

In this main section we develop a bigraphical i/o-type system and prove important properties of it.

Definition 10.7 (Type environment)**.** *A* type environment *(or* typing*) is an unordered finite assignment of types to names, ranged over by* Γ *and* Δ*. The* support $\mathsf{supp}(\Gamma)$ *is the set of names. When regarded as a finite function from names to types we write* $\Gamma(x)$ *for the type assigned to* x *by* Γ*. The* extension *of* Γ *with the assignment* $x : T$ *is denoted* $\Gamma, x : T$ *when* $x \notin \mathsf{supp}(\Gamma)$*. The* disjoint union Γ, Δ *is defined when* $\mathsf{supp}(\Gamma) \cap \mathsf{supp}(\Delta) = \emptyset$*, and is (also) associative and commutative.* $\Gamma_\emptyset$ *denotes the empty typing.*

Definition 10.8 (Syntax of types and typings)**.**

$$V ::= L \mid \bullet \qquad L ::= \#V \mid \mathsf{o}V \mid \mathsf{i}V \qquad \Gamma ::= \Gamma, x : V \mid \Gamma, x : L \mid \Gamma_\emptyset \,.$$

A *link* is a name that may be used for communication. The *values* are the objects (names) that can be communicated along links. The *link types* (L) are the types that can be ascribed to links. The *value types* (V) are the types that can be ascribed to values (names). Link types are value types so that processes can exchange links, allowing mobility. Links can either be used in input $\mathsf{i}V$, output $\mathsf{o}V$, or both $\#V$ (the *connection* type). The inhabitants of unit type $\bullet$ are names. Names assigned type $\bullet$ are *base values* that can only be passed around. There is no special unit value as this would clutter the presentation.

In message-passing process calculi the channels (links) are the essential part because communication is the primitive notion studied. Therefore, in the present paper, we only type links, not nodes. Table 10.2 presents the i/o-typing rules for $\acute{}\mathrm{B}\mathrm{BG}_{\mathrm{sf}\pi}$. The idea is to syntactically define types for

$$\textit{Placings}: \quad \frac{}{\Gamma_\emptyset;\Gamma_\emptyset \vdash \mathsf{1}} \qquad \frac{}{\Gamma_\emptyset;\Gamma_\emptyset \vdash \mathit{join}} \qquad \frac{\Gamma_1 \leq \Gamma_0 \quad \mathsf{supp}(\Gamma_j)^{\,j=0,1} = Z}{\Gamma_0;\Gamma_1 \vdash \gamma_Z}$$

$$\textit{Linkings}: \quad \frac{}{x : L;\Gamma_\emptyset \vdash /x} \qquad \frac{\Gamma \vdash y : T}{X : T;\Gamma \vdash {}^{y}/_{X}}$$

$$\textit{Id \& Conc.}: \quad \frac{\Gamma_1 \leq \Gamma_0 \quad \mathsf{supp}(\Gamma_j)^{\,j=0,1} = \mathit{glob}(I)}{\Gamma_0;\Gamma_1 \vdash \mathsf{id}_I} \qquad \frac{\Gamma_1 \leq \Gamma_0 \quad \mathsf{supp}(\Gamma_j)^{\,j=0,1} = X}{\Gamma_0;\Gamma_1 \vdash \ulcorner X \urcorner}$$

$$\textit{Operators}: \quad \frac{\Delta;\Gamma \vdash b}{\Delta;\Gamma \vdash (X)b} \qquad \frac{\Delta_0;\Gamma_0 \vdash b_0 \;\; \Delta_1;\Gamma_1 \vdash b_1}{\Delta_0,\Delta_1;\Gamma_0,\Gamma_1 \vdash b_0 \otimes b_1} \qquad \frac{\Gamma_0;\Delta \vdash b_0 \;\; \Delta;\Gamma_1 \vdash b_1}{\Gamma_0;\Gamma_1 \vdash b_1 \circ b_0}$$

$$\textit{Ions}: \quad \frac{\Gamma \vdash a : \mathsf{o}T \quad \Gamma' \vdash x : T}{\Gamma_\emptyset;\Gamma,\Gamma' \vdash \mathsf{send}_{ax}} \qquad \frac{\Gamma \vdash a : \mathsf{i}S}{y : S;\Gamma \vdash \mathsf{get}_{a(y)}}$$

$$\textit{Subtyping}: \quad \frac{}{T \leq T} \qquad \frac{S \leq U \quad U \leq T}{S \leq T} \qquad \frac{}{\#T \leq \mathsf{i}T} \qquad \frac{}{\#T \leq \mathsf{o}T}$$

$$\frac{S \leq T}{\mathsf{i}S \leq \mathsf{i}T} \qquad \frac{T \leq S}{\mathsf{o}S \leq \mathsf{o}T} \qquad \frac{T \leq S \quad S \leq T}{\#S \leq \#T}$$

$$\textit{Names}: \quad \frac{S \leq T}{x : S \vdash x : T}$$

Table 10.2: i/o-typing rules for $\acute{}\mathrm{BBG}_{\mathrm{sf}\pi}$.

elementary bigraphs and the operators on them following their structure inductively. The i/o-type system guarantees that ions (images of processes) use their links in accordance with their capabilities on them. The subtyping preorder $\leq$ can be thought of as inclusion between the sets of the values of the types. So we would have, e.g., Int $\leq$ Real. We write $\Gamma_1 \leq \Gamma_0$ whenever $\mathsf{supp}(\Gamma_1) = \mathsf{supp}(\Gamma_0) = X$ and $\forall x \in X.\Gamma_1(x) \leq \Gamma_0(x)$.

Definition 10.9 (Judgments)**.** *A* bigraph type judgment *is of the form* $\Delta;\Gamma \vdash b$*, where* b *is a bigraph term. A* name type judgment *is of the form* $\Gamma \vdash x : T$*.*

Consider Table 10.2. Bigraphs are contexts and thus typed by two typings; a typing Δ of the inner names and one Γ for the outer names. When

assigning types it does not matter whether a name is local or global, we just type the third component of interfaces, which is projected by function *glob*. The type system is *strong* in the sense that the typings carry no information about names that do not appear in the interfaces of the bigraph. The reason for this will become clear later when we treat properties of the type system. In the following we refer to the axioms that govern bigraphical term equality, which are defined in [DB06, Mil05a], and recited in Definition A.32 of Appendix A.1.

Nameless elementary bigraphs, i.e., the barren root and *join*, are typed using two empty typings. Transpositions allow subtyping because they partially coincide with identities (by axioms (C7) $\gamma_{I,\epsilon} = \mathsf{id}_I$ and (C8) $\gamma_{J,I} \circ \gamma_{I,J} = \mathsf{id}_{I \otimes J}$), which in turn partially coincide with substitutions (by axioms (L1) $x/x = \mathsf{id}_x$ and (L3) $/y \circ y = \mathsf{id}_\epsilon$), and substitutions must allow subtyping, see below. We write γ_Z for $\gamma_{m,n,(\vec{X},\vec{Y})}$ when we are merely interested in the names collectively.

A closure – name creation – can be given any link type. Actually, it is only useful when it is a connection type ($\#V$) because for two processes to communicate over a link one needs to use the link for output and the other for input, simultaneously. Bigraphical substitutions y/X demand that all $x \in X$ have the same type T (denoted by $X : T$), which is natural because substituting in a y for any x really identifies these x_j, namely they are y from the viewpoint of the context. In harmony with the i/o-subtyping discipline we must be able to assign to y a subtype of the x_j so as to allow substitution of names with a possibly smaller (more general) capability. In a sense this corresponds to the usual substitution lemma for π-calculi.

Identities and concretions allow subtyping. Concretions merely globalise outer names but are allowed to subtype. This enables a Narrowing lemma.

Localising names does not affect types so the rule for abstraction is straightforward. The rule for tensor product splits the typing in its two branches according to the names of each tensor operand. The rule for composition demands the types of the common interface to be identical, which is natural when considering that bigraphs are really categorical morphisms between objects (interfaces).

The rules for ions are essential as they type the prefixes. Channels are forced to be of output and input type, respectively. Notice the asymmetry between how x and y are typed; the type of y is fixed in the inner typing because it is a binder, just like the cases for the inner typings of closure and substitution.

The rule for names encompasses subsumption, because the typings are

strong rendering obsolete the need to have two separate rules.

The type system is not entirely generic. The following rules are independent from the particular i/o-subtyping discipline: Barren root, *join*, name closure, abstraction, tensor, and composition. The rules for transpositions, substitutions, identities, and concretions all have a subtyping condition, which would probably change depending on the particular typing discipline under consideration. (Nevertheless, the rule for substitution seems safe as it does not allow the elements of X to hve different types, nor to have different types w.r.t y.) Obviously, the rules for subtyping are not generic. The rules for ions are dependent on the particular discipline, and it seems unrealistic to have (entirely) generic ones, because ions represent particular calculus constructs.

This type system is, however, tailored toward message-passing process calculi. Probably, a bigraphical type system for, e.g., Mobile Ambients will look quite different, because there we wish to type controls in hierarchies and not so much links.

This type system differs from traditional type systems, e.g., the i/o-type system of π-calculus (see e.g. [SW01]) in the following respects: 1) We type contexts, not terms, and therefore we have to account for (categorical) composition. 2) Explicit substitution y/x is a syntactic term and hence needs to be typed. This fundamental difference is important because it pervades the properties of the type system in that subtyping of substitution in a sense represents a substitution lemma. 3) The tensor product is more fundamental than parallel product. 4) There is a distinction between local and global names. An important insight is that a name $x \in \mathit{glob}(\mathit{dom}(b))$ and *another* $x \in \mathit{glob}(\mathit{cod}(b))$) are really two different names if they are not linked in b.

The type system enjoys two crucial properties; subject reduction and type soundness. These results rest upon the Main Lemma establishing that the bigraphical typing relation is closed under bigraph *term* equality, which in turn requires *Narrowing* and *Widening*. The typing and subtyping relations enjoy *Inversion*, i.e., can be read "bottom-up", because they are syntax-directed.

Lemma 10.10 (Narrowing)**.** *If* $\Delta; \Gamma, x : T \vdash b$ *and* $S \leq T$ *then* $\Delta; \Gamma, x : S \vdash b$.

Lemma 10.11 (Widening)**.** *If* $\Delta, x : S; \Gamma \vdash b$ *and* $S \leq T$ *then* $\Delta, x : T; \Gamma \vdash b$.

Widening is unusual (for process calculi) in that it is defined on contexts. It is in a sense the dual lemma to Narrowing as it allows widening of inner typings.

The congruence relation = of the Main Lemma is the involved, axiomatised bigraph equality on terms (see [DB06, Mil05a]).The Main Lemma states that if two bigraph terms are equal then they can be typed in the same environments so the type system is robust w.r.t. bigraph equality: term equality on bigraphs coincides with graph isomorphism so this lemma allows us to think of types on the underlying graphs. This lemma is (technically) crucial, and it is unusual because it works for contexts (and not just terms).

Lemma 10.12 (Main Lemma)**.** *Suppose $b_0 = b_1$ for any bigraphs b_0 and b_1. Then $\Delta;\Gamma \vdash b_0$ if and only if $\Delta;\Gamma \vdash b_1$.*

Corollary 10.13 of the Main Lemma tells us that the type system is robust w.r.t. decomposition of the term as a graph, which is important for Subject Reduction.

Corollary 10.13 (Decompositionality)**.** *If $\Delta;\Gamma \vdash b$ and $b = b_1 \circ b_0$ then there exists a typing Θ such that $\Delta;\Theta \vdash b_0$ and $\Theta;\Gamma \vdash b_1$.*

Before stating and proving a subject reduction theorem we consider the grounded rules generated by the parametric rule of Definition 10.4, because the type derivations of this rule's redex and reactum are a key to understanding the proof of the subject reduction theorem. The generated ground rules are of form (r, r'):

$$
\begin{aligned}
& \Big((\mathsf{id}_X \otimes R) \circ d \;,\; (\mathsf{id}_X \otimes R') \circ \varrho(d)\Big) \\
\stackrel{\text{def}}{=}\; & \Big((\mathsf{id}_X \otimes (\mathsf{send}_{ax} \mid \mathsf{get}_{a(y)})) \circ d \;,\; (\mathsf{id}_X \otimes ((\mathsf{id}_1 \mid \mathsf{id}_1 \;\lhd\; {}^{x}\!/_{y}) \;\lhd\; a)) \circ \varrho(d)\Big) \\
\stackrel{\text{def}}{=}\; & \Big((\mathsf{id}_X \otimes ((\mathit{join} \otimes \mathsf{id}_{(ax)}) \circ (\sigma \circ (\mathsf{send}_{ax} \otimes (\tau \circ \mathsf{get}_{a(y)}))))) \circ d \;, \\
& \quad (\mathsf{id}_X \otimes ((\mathit{join} \otimes \mathsf{id}_{(ax)}) \circ (((\mathit{join} \otimes \mathsf{id}_{(x)}) \circ (\mathsf{id}_1 \otimes {}^{(x)}\!/_{(y)})) \otimes (a)))) \circ \varrho(d)\Big)
\end{aligned}
$$

where $\{a, x\} \cap X = \emptyset$, $\tau = {}^{(a')}\!/_{(a)}$, and $\sigma = {}^{(a)}\!/_{(\{a,a'\})} \otimes {}^{(x)}\!/_{(x)}$ w.l.o.g. We remark that ${}^{(x)}\!/_{(y)} \stackrel{\text{def}}{=} (x)({}^{x}\!/_{y} \otimes \mathsf{id}_1) \circ \ulcorner y \urcorner$. Subject Reduction (Theorem 10.14) is the main theorem and guarantees that typings are preserved over reaction. The core in the proof of the theorem is an analysis of redex and reactum as the type derivations of the context, and in this case also the parameters, are preserved by reaction. Hence, the theorem is really a property of reaction rules. For a BRS with multiple (possibly overlapping) reaction rules one would analyse the redex-reactum pair of each one and then simply combine the results to obtain the theorem.

Theorem 10.14 (Subject Reduction)**.** *For process bigraphs b_0 and b_1, if $\Gamma_\emptyset;\Delta \vdash b_0$ and $b_0 \rightarrowtriangle b_1$ then $\Gamma_\emptyset;\Delta \vdash b_1$.*

Proof. The proof is by analysis of the derivation of $b_0 \rightarrowtriangle b_1$ by the sole reaction rule. Because $b_0 \rightarrowtriangle b_1$, then by Definition 10.1 there exists an active context D such that $b_0 = D \circ r$ and $b_1 \eqsim D \circ r'$. Assume a derivation of $\Gamma_\emptyset; \Delta \vdash b_0$, then also $(*)$ $\Gamma_\emptyset; \Delta \vdash D \circ r$ by Lemma 10.12. By Inversion we must have (among others) the following six subderivations from $(*)$: $(1')$ $\Gamma_\emptyset; \Gamma, y : S \vdash d$, $(3')$ $a : U \vdash a : \mathsf{o}T$, $(3'')$ $x : U' \vdash x : T$, $(4')$ $a : R \vdash a : \mathsf{i}S$, $(5')$ $U \leq R$, and $(8')$ $\Gamma', a : W, x : W'; \Delta \vdash D$. We also know that $W \leq U$ and $W' \leq U'$. By $(3')$, $(5')$ and $(4')$ we conclude $T \leq S$ (cf. [SW01]). $W' \leq U'$, and by $(3'')$ we have $U' \leq T \leq S$, so $W' \leq S$.

Now, consider the derivation to be built. $b_1 \eqsim D \circ r'$ implies that $b_1 = D \circ r'$ abstractly. By Lemma 10.12 it suffices to derive $\Gamma_\emptyset; \Delta \vdash D \circ r'$. Reuse the derivation of D. $\varrho = \mathsf{id}_2$ so $\varrho(d) = d$. This means that we can also reuse $(1')$. We still need to justify a derivation of $y : S; x : W' \vdash {}^{(x)}/_{(y)}$. This merely requires justification of $W' \leq S$ because we may choose not to subtype in the other substitutions. $W' \leq S$ has already been established so we can build the desired derivation of $\Gamma_\emptyset; \Delta \vdash D \circ r'$. □

We remark that the inner typings are preserved because the reaction rule is linear, but that need not be the case in general, where the theorem could instead relate the inner typings by something weaker than equality (since sites, including local inner names, can be discarded or replicated and renamed).

Type Soundness (Proposition 10.15) states that a process bigraph b well-typed in $\Gamma_\emptyset; \Gamma$ can only perform input or output actions for which Γ offers the appropriate capabilities.

Proposition 10.15 (Type Soundness)**.** *Let $\rightarrowtriangle^*$ be the reflexive, transitive closure of $\rightarrowtriangle$. Suppose that process bigraph $b = [\![P]\!]_{(X)}$, $\Gamma_\emptyset; \Gamma \vdash b$, and $b \rightarrowtriangle^* b'$. Then, for each non-idle $a \in glob(cod(b'))$ it holds that:*

1. *If $\Gamma \vdash a : \mathsf{i}S$ then a is either linked to the channel port of a* get *ion or linked to the datum port of a* send *ion.*
2. *If $\Gamma \vdash a : \mathsf{o}T$ then a is linked to a* send *ion.*

Proof (Sketch). The proof is by induction on the length of the reduction $b \rightarrowtriangle^* b'$. Essentially, one follows the link map in the the proof. The inductive case uses Subject Reduction. □

Type Soundness gives guarantees about outer names, but not closed names because edges have no type. To achieve a stronger type soundness property – such as "well-typed processes do not reduce to *wrong*" – one

could introduce a tagged version of the BRS in which each name is permanently tagged with the intended i/o usage, like in [PS96] for π-calculus. Or, we could follow [BS06] and type edges to possibly obtain a result of intermediate strength.

Idle names are merely the residue of reaction in Bigraphs so adding or removing them corresponds, in a precise way to be shown below, to Weakening and Strengthening of a type system in π-calculus. Adding and removing idle names actually changes the bigraph (a context) because the codomain changes. These different bigraphs should however correspond to the same source calculus term because they only differ up to names that do not occur in the source term.

Lemma 10.16 (Weakening)**.** *If* $\Delta;\Gamma \vdash b$ *and* $x \notin \mathsf{supp}(\Gamma)$ *then* $\Delta;\Gamma, x:T \vdash b \otimes (x)$.

Lemma 10.17 (Strengthening)**.** *If* $\Delta;\Gamma, x:T \vdash b \otimes (x)$ *then* $\Delta;\Gamma \vdash b$.

Even though these two properties on the surface appear different from those of π-calculi they really do correspond to the usual properties of typed sfπ.

With these important properties in hand it is time to transfer them to the i/o-typed source calculus sfπ. The standard way to map an untyped process calculus into Bigraphs is to consider a trivial type system for the process calculus with just a single type and map derivations of form $\Gamma \vdash P : \diamond$ (see e.g. [SW01]) to the (untyped) bigraph $[\![P]\!]_{(X)}$ exactly when $\mathsf{fn}(P) \subseteq X = \mathsf{supp}(\Gamma)$. The choice $X = \mathsf{supp}(\Gamma)$ coerces a connection between a process bigraph and its (outer) typing. Names in $\mathsf{supp}(\Gamma) \setminus \mathsf{fn}(P)$ become idle names in $[\![P]\!]_{(X)}$ by the translation, recalling that $[\![\mathbf{0}]\!]_{(X)} = (X)$. This is made precise by Lemma 10.18.

Lemma 10.18. *Suppose* $b = [\![P]\!]_{(X)}$ *and* $\Gamma_\emptyset;\Gamma \vdash b$ *with* $\mathsf{fn}(P) \subseteq X = \mathsf{supp}(\Gamma)$. *Then* $[\![P]\!]_{\mathsf{supp}(\Gamma)} \otimes (x) = [\![P]\!]_{(\mathsf{supp}(\Gamma), x:T))}$ *for any type* T.

Using Lemma 10.18 we conclude: $[\![P]\!]_{(X)} \otimes (x) = [\![P]\!]_{(\mathsf{supp}(\Gamma))} \otimes (x) = [\![P]\!]_{(\mathsf{supp}(\Gamma), x:T))}$ for any type T. Then, by Lemma 10.12 we have that $\Delta;\Gamma, x : T \vdash [\![P]\!]_{(\mathsf{supp}(\Gamma))} \otimes (x)$ if and only if $\Delta;\Gamma, x:T \vdash [\![P]\!]_{(\mathsf{supp}(\Gamma), x:T))}$.

We can "read back" the typing rules over the term translation (to be made precise shortly), and thus also the properties of the type system, including Weakening and Strengthening by courtesy of Lemma 10.18. We read back typing rules as follows: The rules for the inactive process and input prefix are straightforward; restriction is type annotated in sfπ using its premise; parallel composition is derived from tensor and composition; the case for output prefix has the twist that in sfπ the typings used to type

the two channels should be the same; and split the rule for names into a rule for names and one for subsumption. Recall that typings in typed sfπ are not strong. Hence, we recover exactly the fragment of the well-known Pierce-Sangiorgi i/o-type system for the π-calculus [PS96, SW01] (see Table A.1 of Appendix A.3). Proposition 10.19 precisely relates the bigraphical type derivations with the ones for sfπ.

Proposition 10.19 (Transfer of Type Derivations)**.** $\Gamma \vdash P : \diamond$ *if and only if* $\Gamma_\emptyset; \Gamma \vdash [\![P]\!]_{(X)}$ *when* $\mathsf{fn}(P) \subseteq X = \mathsf{supp}(\Gamma)$.

Proof (Sketch). The proof is by structural induction on P using Lemmas 10.18 and 10.12. □

The proof is naturally by structural induction on P because we follow the translation of terms when transferring type derivations. We remark that to extend a BRS to accommodate a new source calculus operator one encodes it, and for a new process construct (e.g. a prefix) one adds an ion to the BRS, encodes the extended source calculus in the extended BRS, and finally one gives a typing rule for this new ion. In conclusion: All of the bigraphical properties are transferable.

10.4 Conclusion

We have demonstrated a novel and uniform approach for developing type systems for (process) calculi, through Bigraphs. Type systems are defined inductively over the structure of elementary bigraphs and their operators, as opposed to using a sorting [BS06]. Thus, a computer may possibly verify that a typed term is well-typed. Concretely, we have illustrated the approach by developing a sound i/o-type system enjoying a general form of Weakening and Strengthening for a bigraphical model of a core π-calculus, and then we have transferred the type system and its properties to the π-calculus. The development of the i/o-type system for Bigraphs differs significantly from i/o-typed π-calculus: bigraphs are contexts with richer structure than ordinary process calculus terms, which is reflected in the axioms governing bigraphical term equality, leading to technical intricacies in the Main Lemma used in Subject Reduction; Weakening and Strengthening of typed π-calculi corresponds to adding and removing idle names of bigraph terms, respectively.

We have tackled the case of i/o-types for the π-calculus because, being non-trivial and well-studied, this type system seemed to be an ideal test for

our programme. In the future we would like to consider more sophisticated type systems. Here, some of the potential advantages of bigraphs (in particular, their modularity, the possibility of transferring the type results to a family of concrete calculi, and the insights gained on the type systems themselves) could be particularly valuable. A good example of this might be type systems for deadlock-freedom and lock-freedom, such as Kobayashi and co-workers' [Kob06, Kob02]. These type systems yield fundamental behavioural guarantees on processes such as absence of deadlock. However, one may argue that they are not fully understood yet, as a number of variations have appeared, with different expressive power. Also, they seem very sensitive to the grammar of the underlying process language, so transferring them to a different formalism may be troublesome. Formulating these types at the more abstract level of Bigraphs could shed light into their design and facilitate their application.

We would like also to: consider different process languages, for instance with primitives for distribution such as Mobile Ambients or Homer; further investigate the relation between our work and sortings; generalise our approach to capture several interesting type systems simultaneously; to automatically derive an LTS for the BRS and then lift Subject Reduction to that semantics (to help bridge prior efforts in Bigraphs concerning expressiveness and derivation of LTSs with our approach); and support tools for type inference and type checking.

10.4.1 Additional future work

Moreover, one might: 1) consider i/o types for full(er) π-calculus, and 2) identify "safe" conditions for adding ions and typing rules to the core BRS.

Ad. 1) Let us first consider π-calculus with summation and replication: [Jen07] contains a bigraphical model of π-calculus with summation and replicated reception. The bigraphical model of this calculus given in [Jen07] requires A) *outward binding*, i.e., binders that have scope in sibling nodes (instead of just in child nodes), B) structural congruence in the model is different in that the inactive process is not neutral with respect to parallel composition, and restriction can not be pushed past prefixes nor be eliminated when applied to the inactive process, and C) restriction process bigraphs are removed from the underlying category (because they are not used in the encoding given in [Jen07]). It is thus not clear how to extend our framework to encompass this calculus. No models of other operators nor π-like calculi exist, as far as we know, and it is therefore unclear how to extend our framework to encompass these operators (e.g. matching/mismatching)

and calculi.

Encompassing *asynchronous* π-calculus is straightforward; just add an atomic control representing the new prefix to the signature of $\acute{}\mathrm{B}_{\mathrm{BG}_{\mathrm{sf}\pi}}$, the bigraphical typing rule would be the same, whereas the π-calculus typing rule would just omit the premise $\Gamma \vdash P : \diamond$.

Ad. 2) When adding a new operator to a source calculus one must also supply the translation of it. As long as the already defined operators on elementary bigraphs are enough to capture the semantics then the results will continue to hold. However, when adding a process construct a new ion and a new typing rule must be given, which may break some results. It is not necessarily easier to check that a rule does not break results in the bigraphical type system, but once done the typing rule can be guessed for the encodable calculi and the results hold (provided that Proposition 10.19 can be shown in each case).

Acknowledgments

The first author wishes to thank Mikkel N. Bundgaard, Søren Debois and Troels C. Damgaard for useful technical discussions. We thank the anonymous referees for suggestions on improving this paper's presentation.

This work was funded in part by the Danish Research Agency (grants no.: 2059-03-0031 and 274-06-0415) and the ITU (the LaCoMoCo/BPL and CosmoBiz projects).

10.A Full proofs

This section contains the full proofs.

Lemma 10.20 (Narrowing)**.** *If* $\Delta; \Gamma, x : T \vdash b$ *and* $S \leq T$ *then* $\Delta; \Gamma, x : S \vdash b$.

Proof. The proof is by induction on the height of the derivation of $\Delta; \Gamma, x : T \vdash b$.

- The cases for $\mathsf{1}$, *join*, and $/x$ are vacuously true.
- The cases for transpositions, identities, and concretions hold by Inversion and transitivity of the subtyping relation $\leq$.
- The case for substitutions: Assume a derivation of $X : T'; \Gamma, x : T \vdash y/X$ with premise $\Gamma, x : T \vdash y : T'$ by Inversion. Because typings are strong we must have $\Gamma =_{\Gamma\ \emptyset}$ and $y = x$. Thus, we have a derivation of $y : T \vdash y : T'$. Clearly, $T \leq T'$. By transitivity of subtyping and the

assumption $S \leq T$ we obtain $S \leq T'$. Hence, by the the rule for names (subsumption) we can derive $y : S \vdash y : T'$, which is required to derive $X : T'; \Gamma, x : S \vdash y/x$.

- The case for abstraction follows immediately from Inversion and the induction hypothesis.

- The case for output, send_{az}: Either $x = a$ or $x = z$ but not both. Case $x = a$: Assume a derivation of $\Gamma_\emptyset; a : T, \Gamma \vdash \mathsf{send}_{az}$ with premises (1) $a : T \vdash a : \mathsf{o}T'$ and (2) $\Gamma \vdash x : T'$ by Inversion. From (1) we know that $T \leq \mathsf{o}T'$. Then, the assumption $S \leq T$ and transitivity of subtyping yield $S \leq \mathsf{o}T'$. Then, by subsumption we derive (1′) $a : S \vdash a : \mathsf{o}T'$. Using (1′) and (2) we derive $\Gamma_\emptyset; a : S, \Gamma' \vdash \mathsf{send}_{az}$ as required. The case for $x = z$ is analogous to the case where $x = a$.

- The case for input: The proof is analogous to the proof for output where $x = a$.

- The case for tensor product: Assume a derivation of $\Delta_0, \Delta_1; \Gamma_0, \Gamma_1 \vdash b_0 \otimes b_1$ with premises $\Delta_0; \Gamma_0 \vdash b_0$ and $\Delta_1; \Gamma_1 \vdash b_1$ by Inversion. Either $x \in \mathsf{supp}(\Gamma_0)$ or $x \in \mathsf{supp}(\Gamma_1)$ but not both. In either case the *deciderata* follows from the induction hypothesis on $\Delta_i; \Gamma_i \vdash b_i$.

- The case for composition: Assume a derivation of $\Gamma_0; \Gamma_1 \vdash b_1 \circ b_0$ with premises (1) $\Gamma_0; \Delta \vdash b_0$ and (2) $\Delta; \Gamma_1 \vdash b_1$ by Inversion. We have that $x \in \mathsf{supp}(\Gamma_1)$ so the *deciderata* follows by induction hypothesis on (2).

□

Lemma 10.21 (Widening)**.** *If* $\Delta, x : S; \Gamma \vdash b$ *and* $S \leq T$ *then* $\Delta, x : T; \Gamma \vdash b$.

Proof. The proof is by induction on the height of the derivation of $\Delta, x : S; \Gamma \vdash b$.

- The cases for $\mathsf{1}$, *join*, and output hold vacuously.

- The cases for transpositions, identities, and concretions hold by transitivity of subtyping.

- The case for closure is by axiom.

- The case for substitutions: Assume a derivation of $X : S; \Gamma \vdash y/x$ with premise $\Gamma \vdash y : S$ by Inversion. Clearly, $\Gamma(y) \leq S$, and because $S \leq T$ we obtain $\Gamma(y) \leq T$ by transitivity of subtyping, which derives

$X : T; \Gamma \vdash y/x$ by subsumption. Hence, we can derive $X : T; \Gamma \vdash y/x$ as required.

- The cases for abstraction, tensor, and composition are by Inversion and then one application of the induction hypothesis.
- The case for input: Assume a derivation of $y : S; \Gamma \vdash \mathsf{get}_{a(y)}$ with premise $\Gamma \vdash a : \mathsf{i}S$ by Inversion. $S \leq T$ derives $\mathsf{i}S \leq \mathsf{i}T$ by covariance of subtyping on input types. Clearly, $\Gamma(a) \leq \mathsf{i}S$, and because $\mathsf{i}S \leq \mathsf{i}T$ we obtain $\Gamma(a) \leq \mathsf{i}T$ by transitivity of subtyping. We can thus derive $\Gamma \vdash a : \mathsf{i}T$ and then conclude $y : T; \Gamma \vdash \mathsf{get}_{a(y)}$ as required.

□

Lemma 10.22 (Main Lemma)**.** *Suppose $b_0 = b_1$ for any bigraphs b_0 and b_1. Then $\Delta; \Gamma \vdash b_0$ if and only if $\Delta; \Gamma \vdash b_1$.*

Proof. The proof is by induction on the height of the derivation of $b_0 = b_1$ and has a case for each axiom. The proof consists of 60 cases; there is one case for each direction of each axiom, casing on whether the ion is send or get, and checks for reflexivity, symmetry, transitivity, and congruence. Inversion is used frequently in a straightforward manner so we omit explicit mention of it. We proceed by case analysis.

- case $A \circ \mathsf{id}_I = A$ for $A : I \to J$.

  "$\Rightarrow$": Assume a derivation of $\Delta; \Gamma \vdash A \circ \mathsf{id}_I$ with premises (1) $\Delta; \Theta \vdash \mathsf{id}_I$ and (2) $\Theta; \Gamma \vdash A$. From (1) we know that $\Theta \leq \Delta$ so by Widening on (2) we obtain $\Delta; \Gamma \vdash A$ as required.

  "$\Leftarrow$": Assume a derivation of (1) $\Delta; \Gamma \vdash A$. Clearly, we may directly derive (2) $\Delta; \Delta \vdash \mathsf{id}_I$ by the rule for identities. Now we can build the required derivation of $\Delta; \Gamma \vdash A \circ \mathsf{id}_I$ by the rule for composition using (2) and (1).

- case $A = \mathsf{id}_J \circ A$ for $A : I \to J$.

  "$\Rightarrow$": Assume a derivation of (1) $\Delta; \Gamma \vdash A$. Clearly, we may directly derive (2) $\Gamma; \Gamma \vdash \mathsf{id}_J$ by the rule for identities. Now we can build the required derivation of $\Delta; \Gamma \vdash \mathsf{id}_J \circ A$ by the rule for composition using (1) and (2).

  "$\Leftarrow$": Assume a derivation of $\Delta; \Gamma \vdash \mathsf{id}_J \circ A$ with premises (1) $\Delta; \Theta \vdash A$ and (2) $\Theta; \Gamma \vdash \mathsf{id}_J$. From (2) we know that $\Gamma \leq \Theta$ so by Narrowing on (1) we obtain $\Delta; \Gamma \vdash A$.

- case $A \circ (B \circ C) = (A \circ B) \circ C$.

  "$\Longleftrightarrow$": Clearly, because exactly the same subderivations are needed in both derivations, and the disjoint union on typings is associative.

- case $A \otimes \mathsf{id}_\epsilon = A$.

  "$\Rightarrow$": Reuse the subderivation $\Gamma; \Delta \vdash A$.

  "$\Leftarrow$": Reuse the subderivation $\Gamma; \Delta \vdash A$, and $\Gamma_\emptyset; \Gamma_\emptyset \vdash \mathsf{id}_\epsilon$ is by axiom.

- case $A = \mathsf{id}_\epsilon \otimes A$.

  Analogous to the previous case.

- case $A \otimes (B \otimes C) = (A \otimes B) \otimes C$.

  Clearly, because exactly the same subderivations are needed in both derivations, and the disjoint union on typings is associative.

- case $\mathsf{id}_I \otimes \mathsf{id}_J = \mathsf{id}_{I \otimes J}$.

  "$\Rightarrow$": Assume a derivation of $\Delta_0, \Delta_1; \Gamma_0, \Gamma_1 \vdash \mathsf{id}_I \otimes \mathsf{id}_J$ with premises (1) $\Delta_0; \Gamma_0 \vdash \mathsf{id}_I$ and (2) $\Delta_1; \Gamma_1 \vdash \mathsf{id}_J$, where $\mathsf{supp}(\Delta_0) = \mathsf{supp}(\Gamma_0) = \mathit{glob}(I)$ and $\Gamma_0 \leq \Delta_0$, and $\mathsf{supp}(\Delta_1) = \mathsf{supp}(\Gamma_1) = \mathit{glob}(J)$ and $\Gamma_1 \leq \Delta_1$. We need to establish (A) $\mathsf{supp}(\Delta_0, \Delta_1) = \mathsf{supp}(\Gamma_0, \Gamma_1) = \mathit{glob}(I \otimes J)$ and (B) $\Gamma_0, \Gamma_1 \leq \Delta_0, \Delta_1$. We know that $\mathsf{supp}(\Delta_0) = \mathit{glob}(I)$ and $\mathsf{supp}(\Delta_1) = \mathit{glob}(J)$ so we obtain that $\mathsf{supp}(\Delta_0, \Delta_1) = \mathit{glob}(I \otimes J)$. Likewise for Γ_0 and Γ_1 w.r.t J so (A) is established. We know that $\Gamma_k \leq \Delta_k$ (for $k = 0, 1$) so clearly $\Gamma_0, \Gamma_1 \leq \Delta_0, \Delta_1$, which establishes (B). We can now derive $\Delta_0, \Delta_1; \Gamma_0, \Gamma_1 \vdash \mathsf{id}_{I \otimes J}$ as required.

  "$\Leftarrow$": Assume a derivation of $\Delta; \Gamma \vdash \mathsf{id}_{I \otimes J}$ with premises (1) $\mathsf{supp}(\Delta) = \mathit{glob}(I \otimes J)$, (2) $\mathsf{supp}(\Gamma) = \mathit{glob}(I \otimes J)$, and (3) $\Gamma \leq \Delta$ by Inversion. Clearly, Δ and Γ can be split into parts Δ_0, Δ_1 and Γ_0, Γ_1 such that $\mathsf{supp}(\Theta_0) = \mathit{glob}(I)$ and $\mathsf{supp}(\Theta_1) = \mathit{glob}(J)$ (for $\Theta \in \{\Delta, \Gamma\}$). It is also obvious that $\Gamma_0 \leq \Delta_0$ and $\Gamma_1 \leq \Delta_1$. Hence, we can derive $\Delta_0; \Gamma_0 \vdash \mathsf{id}_I$ and $\Delta_1; \Gamma_1 \vdash \mathsf{id}_J$, and thus also $\Delta; \Gamma \vdash \mathsf{id}_I \otimes \mathsf{id}_J$ as required.

- case $(A_1 \otimes B_1) \circ (A_0 \circ B_0) = (A_1 \circ A_0) \otimes (B_1 \circ B_0)$.

  "$\Longleftrightarrow$": Clearly, because exactly the same subderivations are needed in both derivations, albeit slightly rearranged.

- case $\gamma_{I,\epsilon} = \mathsf{id}_I$.

  Recall that $\gamma_{I,\epsilon} \overset{\text{def}}{=} \gamma_{m,0,(\vec{X}_B,())} \otimes \mathsf{id}_{X_F} \otimes \mathsf{id}_\emptyset \overset{\text{def}}{=} \gamma_{m,0,(\vec{X}_B,())} \otimes \mathsf{id}_{X_F}$ for $I = \langle m, \vec{X}_B, \{\vec{X}_B\} \uplus X_F \rangle$ w.l.o.g.

"⇒": Assume a derivation of $\Delta_0, \Delta_1; \Gamma_0, \Gamma_1 \vdash \gamma_{m,0,(\vec{X}_B,())} \otimes \mathsf{id}_{X_F}$ with premises $\Delta_0; \Gamma_0 \vdash \gamma_{m,0,(\vec{X}_B,())}$ and $\Delta_1; \Gamma_1 \vdash \mathsf{id}_{X_F}$. From these premises we clearly have $\mathsf{supp}(\Delta_0, \Delta_1) = \mathsf{supp}(\Gamma_0, \Gamma_1) = \{\overrightarrow{X_B}\} \uplus X_F = glob(I)$, and also $(\Gamma_0, \Gamma_1)(i) \leq (\Delta_0, \Delta_1)(i)$ for any $i \in glob(I)$. Thus, a derivation of $\Delta_0, \Delta_1; \Gamma_0, \Gamma_1 \vdash \mathsf{id}_I$ can be built.

"⇐": Assume a derivation of $\Gamma_0; \Gamma_1 \vdash \mathsf{id}_I$ with premises $\mathsf{supp}(\Gamma_j) = glob(I)$ (for $j = 0, 1$) and $\Gamma_1 \leq \Gamma_0$. $glob(I) = \{\overrightarrow{X_B}\} \uplus X_F$ by assumption. Clearly, Γ_j can be split into typings Γ_j for $\{\vec{X}_B\}$ and Γ'_j for X_F. Therefore, the rule for tensor can be used to build the required derivation of $\Gamma_0, \Gamma'_0; \Gamma_1, \Gamma'_1 \vdash \gamma_{m,0,(\vec{X}_B,())} \otimes \mathsf{id}_{X_F}$.

- $\gamma_{J,I} \circ \gamma_{I,J} = \mathsf{id}_{I \otimes J}$.

  Recall that $\gamma_{J,I} \stackrel{\text{def}}{=} \gamma_{n,m,(\vec{Z}_B,\vec{X}_B)} \otimes \mathsf{id}_{Z_F} \otimes \mathsf{id}_{X_F}$ and $\gamma_{I,J} \stackrel{\text{def}}{=} \gamma_{m,n,(\vec{X}_B,\vec{Z}_B)} \otimes \mathsf{id}_{X_F} \otimes \mathsf{id}_{Z_F}$ for $I = \langle m, \vec{X}_b, \{\vec{X}_B\} \uplus X_F \rangle$ and $J = \langle n, \vec{Z}_b, \{\vec{Z}_B\} \uplus Z_F \rangle$ w.l.o.g.

  "⇒": Clearly, $\gamma_{J,I} \circ \gamma_{I,J}$ is typable in some Γ_j (for $j = 0, 1$) assigning types to exactly the names of I and J, disjointly, with $\Gamma_1 \leq \Gamma_0$. Thus, a derivation of $\Gamma_0; \Gamma_1 \vdash \mathsf{id}_{I \otimes J}$ can be built.

  "⇐": Reverse the argument.

- case $\gamma_{I,J} \circ (A \otimes B) = (B \otimes A) \circ \gamma_{H,J}$ for $A : H \to I$, $B : J \to K$.

  "⇒": Assume a derivation of $\Gamma_0; \Gamma_1 \vdash \gamma_{I,J} \circ (A \otimes B)$, where $\mathsf{supp}(\Gamma_0) = glob(H) \uplus glob(J)$ and $\mathsf{supp}(\Gamma_1) = glob(I) \uplus glob(K)$. Exactly the same subderivations are needed to build a derivation of $(B \otimes A) \circ \gamma_{H,J}$, albeit slightly rearranged.

  "⇐": The argument is analogous.

- case $\gamma_{I \otimes J,K} = (\gamma_{I,K} \otimes \mathsf{id}_J) \circ (\mathsf{id}_I \otimes \gamma_{J,K})$.

  Essentially, this case holds because exactly the same names are typed on both sides, albeit somewhat rearranged.

  Recall that $\gamma_{I \otimes J,K} \stackrel{\text{def}}{=} \gamma_{(I \otimes J)_B,K_B} \otimes \mathsf{id}_{(I \otimes J)_F} \otimes \mathsf{id}_{K_F}$, where the subscripts B and F signify the set of bound/local names and the set of free names of an interface, respectively.

  For any interface I, let I_F denote the free (i.e. $glob(I) \setminus \mathsf{cell}(I)$) and I_B the bound (i.e. $\mathsf{cell}(I)$) names.

  "⇒": Assume a derivation of $\Gamma; \Gamma' \vdash \gamma_{(I \otimes J)_B,K_B} \otimes \mathsf{id}_{(I \otimes J)_F} \otimes \mathsf{id}_{K_F}$ with subderivations (1) $\Gamma_1; \Gamma'_1 \vdash \gamma_{(I \otimes J)_B,K_B}$, (2) $\Gamma_2; \Gamma'_2 \vdash \mathsf{id}_{(I \otimes J)_B}$, and (3) $\Gamma_3; \Gamma'_3 \vdash$

id_{K_F}, where $\Gamma = \Gamma_1, \Gamma_2, \Gamma_3$ and $\Gamma' = \Gamma'_1, \Gamma'_2, \Gamma'_3$, $\mathsf{supp}(\Gamma_1)$. Notice that $\mathsf{supp}(\Gamma_1) = (I \otimes J)_B \uplus K_B$, $\mathsf{supp}(\Gamma_2) = (I \otimes J)_F$, and $\mathsf{supp}(\Gamma_3) = K_F$.

Recall that $\gamma_{J,K} \stackrel{\text{def}}{=} \gamma_{J_B,K_B} \otimes \mathsf{id}_{J_F} \otimes \mathsf{id}_{K_F}$ and $\gamma_{I,K} \stackrel{\text{def}}{=} \gamma_{I_B,K_B} \otimes \mathsf{id}_{I_F} \otimes \mathsf{id}_{K_F}$.

We need to build a derivation of $\Gamma; \Gamma' \vdash (\gamma_{I,K} \otimes \mathsf{id}_J) \circ (\mathsf{id}_I \otimes \gamma_{J,K})$. This requires two subderivations: (A) $\Gamma; \Gamma'' \vdash \mathsf{id}_I \otimes \gamma_{J,K}$ and (B) $\Gamma; \Gamma'' \vdash \gamma_{I,K} \otimes \mathsf{id}_J$. Both (A) and (B) require two subderivations. (A1) $\Gamma_I; \Gamma''_I \vdash \mathsf{id}_I$ and (A2) $\Gamma_{J\otimes K}; \Gamma''_{J\otimes K} \vdash \gamma_{J,K}$, where Γ_I denotes $\Gamma \restriction glob(I)$ for any typing Γ and interface I. The case for (B) is analogous. So, we must find a suiting Γ''. Pick $\Gamma'' = \Gamma''_I, \Gamma''_{J\otimes K} = \Gamma_I, \Gamma_{J\otimes K} = \Big((\Gamma_1 \restriction I_B), \Gamma_2 \restriction I_F\Big), \Big((\Gamma_1 \restriction (J_B \uplus J_K)), (\Gamma_2 \restriction J_F), \Gamma_3\Big)$. All that remains is to check (i) $\Gamma''_I \leq \Gamma_I$ and $\mathsf{supp}(\Gamma_I) = \mathsf{supp}(\Gamma''_I) = glob(I)$, (ii) $\Gamma''_{J\otimes K} \leq \Gamma_{J\otimes K}$ and $\mathsf{supp}(\Gamma_{J\otimes K}) = \mathsf{supp}(\Gamma''_{J\otimes K}) = glob(J) \uplus glob(K)$, and (iii) $\Gamma_I, \Gamma_{J\otimes K} = \Gamma$ and $\Gamma''_I, \Gamma''_{J\otimes K} = \Gamma''$. (i) follows from the fact that $(\Gamma_1 \restriction I_B), \Gamma_2 \restriction I_F = \Gamma_I = \Gamma''_I$. (ii) follows from the fact that $(\Gamma_1 \restriction (J_B \uplus K_B)), (\Gamma_2 \restriction J_F), \Gamma_3 = \Gamma_{J\otimes K} = \Gamma''_{J\otimes K}$. (iii) follows from our choice of $\Gamma'' = \Gamma$.

"$\Leftarrow$": Obviously, we can make the same appropriate splits of typings.

- case $x/x = \mathsf{id}_x$.

  "$\Rightarrow$:" Assume a derivation of $x : T; \Gamma \vdash x/x$ with premise (*) $\Gamma \vdash x : T$. (*) implies (1) $\mathsf{supp}(\Gamma) = \{x\}$ and (2) $\Gamma(x) \leq T$. Also, clearly (3) $\mathsf{supp}(x : T) = \{x\}$. Thus, we can from (1), (2), and (3) build the desired derivation of $x : T; \Gamma \vdash \mathsf{id}_x$.

  "$\Leftarrow$": Assume a derivation of $\Gamma_0; \Gamma_1 \vdash \mathsf{id}_x$ with premises $\mathsf{supp}(\Gamma_i) = \{x\}$ (for $i = 0, 1$) and (*) $\Gamma_1(x) \leq \Gamma_0(x)$. (*) implies that $\Gamma_1 \vdash x : \Gamma_0(x)$, which allows us to conclude $\Gamma_0; \Gamma_1 \vdash x/x$ as required.

- case $/y \circ y/x = /x$.

  "$\Rightarrow$": Assume a derivation of $x : L; \Gamma_\emptyset \vdash /y \circ y/x$ with premises $x : L; y : L' \vdash y/x$ and $y : L'; \Gamma_\emptyset \vdash /y$, for some link types L and L' such that $L' \leq L$. By axiom, $x : L; \Gamma_\emptyset \vdash /x$.

  "$\Leftarrow$": Assume a derivation of $x : L; \Gamma_\emptyset \vdash /x$. Build derivations of $x : L; y : L \vdash y/x$ and $y : L; \Gamma_\emptyset \vdash /y$ by axioms. From these two subderivations we construct a derivation of $x : L; \Gamma_\emptyset \vdash /y \circ y/x$.

- case $/y \circ y = \mathsf{id}_\epsilon$.

  Recall that y is shorthand for $y/\emptyset$.

"⇒:" Assume a derivation of $\Gamma_\emptyset;\Gamma_\emptyset \vdash /y \circ y$ with premises $\Gamma_\emptyset; y : L \vdash y/\emptyset$ and $y : L;\Gamma_\emptyset \vdash /y$. As required, $\Gamma_\emptyset;\Gamma_\emptyset \vdash \mathsf{id}_\epsilon$ is trivially derivable.

"⇐": Assume a derivation of $\Gamma_\emptyset;\Gamma_\emptyset \vdash \mathsf{id}_\epsilon$. To build a derivation of $\Gamma_\emptyset;\Gamma_\emptyset \vdash /y \circ y$ two subderivations are needed: $\Gamma_\emptyset; y : L \vdash y/\emptyset$ and $y : L;\Gamma_\emptyset \vdash /y$, for some link type L. They are both trivially derivable.

- case $z/_{(Y \uplus \{y\})} \circ (\mathsf{id}_Y \otimes y/_X) = z/_{(Y \uplus X)}$.

  "⇒": Assume a derivation of $\Gamma_0, X : T;\Gamma^z \vdash z/_{(Y \uplus \{y\})} \circ (\mathsf{id}_Y \otimes y/_X)$ with three subderivations: (1) $\Gamma_0;\Gamma_1 \vdash \mathsf{id}_Y$, (2) $X : T;\Gamma^y \vdash y/_X$, and (3) $\Gamma_1, \Gamma^y;\Gamma^z \vdash z/_{(Y \uplus \{y\})}$, with the following premises: (1A) $\mathsf{supp}(\Gamma_j) = Y$ (for $j = 0, 1$), (1B) $\Gamma_1 \leq \Gamma_0$, (2A) $\Gamma^y \vdash y : T$, and (3A) $\Gamma^z \vdash z : (\Gamma_1, \Gamma^y)(Y \uplus \{y\})$.

  We need to establish (*) $\Gamma^z \vdash z : (\Gamma_0, X : T)(Y \uplus X)$ to derive the required conclusion $\Gamma_0, X : T;\Gamma^z \vdash z/_{(Y \uplus X)}$. Establishing (*) is obviously equivalent to showing (*1) $\Gamma^z \vdash z : \Gamma_0(Y)$ and (*2) $\Gamma^z \vdash z : T$, because $(X : T)(X) = T$.

  From (1A) we know that $\mathsf{supp}(\Gamma_0) = Y$ so $\mathsf{supp}(\Gamma_0, X : T) = Y \uplus X$. From (3A) we have $\Gamma^z \vdash z : \Gamma_1(Y)$, in particular, so $\Gamma^z(z) \leq \Gamma_1(Y) \leq \Gamma_0(Y)$, by (1B). Thus, by transitivity of subtyping $\Gamma^z(z) \leq \Gamma_0(Y)$ and we can derive (*1) by subsumption.

  For (*2) we must establish that $\Gamma^z(z) \leq T$. From (3A) we know that $\Gamma^z \vdash z : \Gamma^y(y)$ so $\Gamma^z(z) \leq \Gamma^y(y)$. From (2A) we know that $\Gamma^y(y) \leq T$. Combining these two facts and transitivity of subtyping we obtain $\Gamma^z(z) \leq T$. Hence, we can derive (*2) by subsumption.

  Finally, we derive the *deciderata* by (*1) and (*2) and the rule for substitutions.

  "⇐": Assume a derivation of $(Y \uplus X) : T;\Gamma \vdash z/_{(Y \uplus X)}$ with premise (*) $\Gamma \vdash z : T$. We need to show $(Y \uplus X) : T;\Gamma \vdash z/_{(Y \uplus \{y\})} \circ (\mathsf{id}_Y \otimes y/_X)$. Thus, to use the rule for composition we must establish two premises: (1) $(Y \uplus X) : T;\Delta \vdash \mathsf{id}_Y \otimes y/_X$ and (2) $\Delta;\Gamma \vdash z/_{(Y \uplus \{y\})}$, for some suitable Δ. We pick $\Delta = Y : T, y : T$. Then, (2) is derivable if we can establish $\Gamma \vdash z : T$, but this follows from (*). Hence, only (1) remains. We can use the rule for tensor with premises (1A) $Y : T; Y : T \vdash \mathsf{id}_Y$ and (1B) $X : T; y : T \vdash y/_X$, because $Y : T, X : T \stackrel{\text{def}}{=} (Y \uplus X) : T$. (1A) is trivial by the rule for identities and (1B) trivial by the rule for substitutions.

- case $\mathit{join} \circ (1 \otimes \mathsf{id}_1) = \mathsf{id}_1$.

  "⇐⇒": Clearly, $\Gamma_\emptyset;\Gamma_\emptyset$ types both sides by axioms.

- case $\mathit{join} \circ (\mathit{join} \otimes \mathsf{id}_1) = \mathit{join} \circ (\mathsf{id}_1 \otimes \mathit{join})$.

  "$\Longleftrightarrow$": Clearly, $\Gamma_\emptyset; \Gamma_\emptyset$ types both sides by axioms.

- case $\mathit{join} \circ \gamma_{1,1,(\vec{\emptyset},\vec{\emptyset})} = \mathit{join}$.

  "$\Longleftrightarrow$": Clearly, $\Gamma_\emptyset; \Gamma_\emptyset$ types both sides by axioms.

- case $(\emptyset)P = P$.

  "$\Rightarrow$": Assume a derivation of $\Delta; \Gamma \vdash (\emptyset)P$ with premise $\Delta; \Gamma \vdash P$. Reuse the premise on the right-hand side.

  "$\Leftarrow$": Reuse the assumed derivation of $\Delta; \Gamma \vdash P$ as premise in the rule for abstraction.

- case $(Y)\ulcorner Y \urcorner = \mathsf{id}_{(Y)}$.

  Recall that $\mathsf{id}_{(Y)} \stackrel{\text{def}}{=} {}_{(Y)}/_{(Y)} \stackrel{\text{def}}{=} (Y)\Big(((\bigotimes_{i=1}^{n} {}_{y_i}/_{y_i}) \otimes \mathsf{id}_1) \circ \ulcorner Y \urcorner\Big)$ for $Y = \{y_1, \ldots, y_n\}$.

  "$\Rightarrow$": Assume a derivation of $\Gamma_0; \Gamma_1 \vdash (Y)\ulcorner Y \urcorner$ with premise (*) $\Gamma_0; \Gamma_1 \vdash \ulcorner Y \urcorner$. To build a derivation we essentially need two subderivations: (1) $\Gamma_0; \Gamma_1 \vdash \ulcorner Y \urcorner$ and (2) $\Gamma_1; \Gamma_1 \vdash \bigotimes_{i=1}^{n} {}_{y_i}/_{y_i}$, where (1) is simply by (*). (2) follows from n subderivations of form $\Gamma_1 \restriction y_i; \Gamma_1 \restriction y_i \vdash {}_{y_i}/_{y_i}$ (where $\Gamma_1 \restriction y_i$ restricts Γ_1 to y_i), because $\Gamma_1 \restriction y_i \vdash y : (\Gamma_1 \restriction y_i)(y_i)$ is by axiom and reflexivity of subtyping.

  "$\Leftarrow$": Assume a derivation of $\Gamma_0; \Gamma_1 \vdash \mathsf{id}_{(Y)}$ with premises (1) $\Gamma_0; \Gamma_2 \vdash \ulcorner Y \urcorner$ and (2) $\Gamma_1 \restriction y_i \vdash y_i : (\Gamma_2 \restriction y_i)(y_i)$, for some Γ_2 such that $\Gamma_0; \Gamma_2 \vdash \ulcorner Y \urcorner$ and $\Gamma_2; \Gamma_1 \vdash \bigotimes_{i=1}^{n} {}_{y_i}/_{y_i}$. Then, (1) implies that $\Gamma_2 \leq \Gamma_0$ and (2) implies that $\Gamma_1 \leq \Gamma_2$, so $\Gamma_1 \leq \Gamma_0$ by transitivity of subtyping. Hence, we can derive $\Gamma_0; \Gamma_1 \vdash \ulcorner Y \urcorner$ and thus $\Gamma_0; \Gamma_1 \vdash (Y)\ulcorner Y \urcorner$ as required.

- case $(\ulcorner X \urcorner^Z \otimes \mathsf{id}_Y) \circ (X)P = P$ for $P : I \to \langle 1, Z, Z \uplus X \uplus Y \rangle$.

  Recall that $\ulcorner X \urcorner^Z \stackrel{\text{def}}{=} (Z)\ulcorner Z \uplus X \urcorner : \langle 1, Z \uplus X, Z \uplus X \rangle \to \langle 1, Z, Z \uplus X \rangle$.

  "$\Rightarrow$": Assume a derivation of $\Delta; \Gamma \vdash (((Z)\ulcorner Z \uplus X \urcorner) \otimes \mathsf{id}_Y) \circ (X)P$ with subderivations (1) $\Delta; \Gamma' \vdash P$, (2) $\Gamma_0; \Gamma_1 \vdash \ulcorner Z \uplus X \urcorner$, and (3) $\Gamma_2; \Gamma_3 \vdash \mathsf{id}_Y$, with $\Gamma = \Gamma_{\ 1}, \Gamma_2$ and $\Gamma' = \Gamma_0, \Gamma_2$. (2) implies $\Gamma_1 \leq \Gamma_0$ and (3) implies $\Gamma_3 \leq \Gamma_2$, so together they imply $\Gamma = \Gamma_{\ 1}, \Gamma_3 \leq \Gamma_0, \Gamma_2 = \Gamma'$. We have $\Delta; \Gamma' \vdash P$ from (1), and because $\Gamma \leq \Gamma'$ we obtain $\Delta; \Gamma \vdash P$ by Narrowing.

  "$\Leftarrow$": Assume a derivation (*) $\Delta; \Gamma \vdash P$. We can easily build the desired derivation of $\Delta; \Gamma \vdash (((Z)\ulcorner Z \uplus X \urcorner) \otimes \mathsf{id}_Y) \circ (X)P$ as follows: We need the same subderivations as were assumed in the previous case, but by

picking $\Gamma_0 = \Gamma_1$ and $\Gamma_2 = \Gamma_3$ we clearly have subderivations (2) and (3) from above by trivial uses of the rule for concretions and identities, respectively. Thus, we can derive $\Gamma; \Gamma \vdash (Z)^{\ulcorner} Z \uplus X^{\urcorner} \otimes \mathsf{id}_Y$. Finally, we can reuse (*) and by the rule for composition we obtain the *deciderata*.

- case $(((Y)P) \otimes \mathsf{id}_X) \circ G = ((Y)(P \otimes \mathsf{id}_X)) \circ G$.

  "$\Longleftrightarrow$": Clearly, because the subderivations are the same on both sides albeit slightly rearranged, and only the rules for abstraction, tensor and composition are used to build the required derivations.

- case $(X \uplus Y)P = (X)((Y)P)$.

  "$\Longleftrightarrow$": Clearly, because the names are the same on both sides, and the rule for abstraction merely propagates information.

- case $(\mathsf{id}_1 \otimes \alpha) \circ \mathsf{K}_{\vec{y}(\vec{X})} = \mathsf{K}_{\alpha(\vec{y})(\vec{X})}$. Two cases: Either K is send or get.

  - $\mathsf{K}_{\vec{y}(\vec{X})} = \mathsf{send}_{az}$. We have $\alpha \stackrel{\text{def}}{=} {}^{a'}/_{a} \otimes {}^{z'}/_{z}$ for some names a', z where $a' \notin \{z', z\}$ and $z' \notin \{a', a\}$ because α is a renaming, i.e., a bijective substitution. ($a \neq z$ by definition.)
    "$\Rightarrow$": Assume a derivation of $\Gamma_\emptyset; \Gamma \vdash (\mathsf{id}_1 \otimes \alpha) \circ \mathsf{send}_{az}$ with subderivations (A) $\Gamma_\emptyset; \Theta \vdash \mathsf{send}_{az}$ and (B) $\Theta; \Gamma \vdash \mathsf{id}_1 \otimes \alpha$ with $\mathsf{supp}(\Theta) = \{a, z\}$ and $\mathsf{supp}(\Gamma) = \{a', z'\}$. (A) has premises (A1) $\Theta^a \vdash a : \mathsf{o}T$ and (A2) $\Theta^z \vdash z : T$, where Θ^a denotes $\Theta \restriction \{a\}$ and so forth. (B) has premises (B1) $\Gamma^{a'} \vdash a' : \Theta^a(a)$ and (B2) $\Gamma^{z'} \vdash z' : \Theta^z(z)$. Build the desired derivation of $\Gamma_\emptyset; \Gamma \vdash \mathsf{send}_{a'z'}$ with premises (i) $\Gamma^{a'} \vdash a' : \mathsf{o}T$ and (ii) $\Gamma^{z'} \vdash z' : T$. (i) is justified as follows: By (B1), $\Gamma^{a'}(a') \leq \Theta^a(a)$. So, by Narrowing on (A1) we obtain $\Gamma^{a'} \vdash a' : \mathsf{o}T$, bearing in mind that (*) $x : S \vdash x : Tiffy : S \vdash y : T$. (ii) is justified analogously.
    "$\Leftarrow$": Find $\Theta = \Theta^a, \Theta^z$ with $\Theta^a(a) = \Gamma^{a'}(a')$ and $\Theta^z(z) = \Gamma^{z'}(z')$, yielding (B1) and (B2). Then, (A1) follows easily from (i) and (A2 from (ii), because of (*).
  - $\mathsf{K}_{\vec{y}(\vec{X})} = \mathsf{get}_{a(z)}$. We have $\alpha \stackrel{\text{def}}{=} {}^{a}/_{a}$ for some $a' \neq z$ w.l.o.g.
    "$\Rightarrow$": Assume a derivation of $z : S; \Gamma \vdash (\mathsf{id}_1 \otimes {}^{a'}/_{a}) \circ \mathsf{get}_{a(z)}$ with premises (1) $a : U \vdash a : \mathsf{i}S$ and (2) $\Gamma \vdash a' : U$, for some types S and U. (2) implies that $\Gamma(a') \leq U$ so we may apply Narrowing to (1) and obtain $\Gamma \vdash a' : \mathsf{i}S$, because $a : U \vdash a : \mathsf{i}S$ iff $a' : U \vdash a' : \mathsf{i}S$). Then we can build a derivation of $z : S; \Gamma \vdash \mathsf{get}_{a'(z)}$, as required.
    "$\Leftarrow$": Analogous to the "$\Leftarrow$" direction of the send case.

- case $\mathsf{K}_{\vec{y}(\vec{X})} \circ \sigma^{\mathrm{loc}} = \mathsf{K}_{\vec{y}((\sigma^{\mathrm{loc}})^{-1}(\vec{X}))}$. Two cases. Either K is send or get.

  - $\mathsf{K}_{\vec{y}(\vec{X})} = \mathsf{send}_{ax}$. This case holds both ways trivially because send has no local names.
  - $\mathsf{K}_{\vec{y}(\vec{X})} = \mathsf{get}_{a(z)}$. We have $\sigma^{\mathrm{loc}} = {}^{(z)}\!/_{(Z)}$ for some name set Z. Recall that ${}^{(z)}\!/_{(Z)} \stackrel{\mathrm{def}}{=} (z)\Big(({}^{z}\!/_{Z} \otimes \mathsf{id}_1) \circ \ulcorner Z \urcorner\Big)$.

    "$\Rightarrow$:" Assume a derivation of $Z : T; \Gamma \vdash \mathsf{get}_{a(z)} \circ (z)\Big(({}^{z}\!/_{Z} \otimes \mathsf{id}_1) \circ \ulcorner Z \urcorner\Big)$ with premises (1) $S \leq T$ and (2) $\Gamma \vdash a : \mathsf{i}S$. Building a derivation of $Z : T; \Gamma \vdash \mathsf{get}_{a(Z)}$ requires $\Gamma \vdash a : \mathsf{i}T$, which in turn requires (A) $\Gamma \vdash a : \mathsf{i}S$ and (B) $\mathsf{i}S \leq \mathsf{i}T$. (A) follows from (2) and (B) from (1) by the subtyping rule of input types.

    "$\Leftarrow$": The same premises are needed.

- case $b = b$ (reflexivity).

  Show that $\Delta; \Gamma \vdash b$ if and only if $\Delta; \Gamma \vdash b$, but this is immediate.

- case $b = b' \implies b' = b$ (symmetry).

  Read the statement as a rule: $\dfrac{b = b'}{b' = b}$.

  Assume: $b_0 = b_1$. Show: $\Delta; \Gamma \vdash b_0$ if and only if $\Delta; \Gamma \vdash b_1$. We are proceeding by induction on the derivation of $b_0 = b_1$ and the last rule used was the rule for symmetry, so we must have had $b_1 = b_0$. Now, by induction hypothesis on $b_1 = b_0$ we obtain $\Delta; \Gamma \vdash b_1$ if and only if $\Delta; \Gamma \vdash b_0$. Because "iff" is symmetric we have the *desiderata*.

- case $b = b' \,\&\, b' = b'' \implies b = b''$ (transitivity).

  Read the statement as a rule: $\dfrac{b = b' \quad b' = b''}{b = b''}$.

  Assume: $b_0 = b_1$. Show: $\Delta; \Gamma \vdash b_0$ if and only if $\Delta; \Gamma \vdash b_1$. We have must have had $b_0 = b_2$ and $b_2 = b_1$ for some b_2. By induction hypothesis: (1) $\Delta; \Gamma \vdash b_0$ if and only if $\Delta; \Gamma \vdash b_2$, and (2) $\Delta; \Gamma \vdash b_2$ if and only if $\Delta; \Gamma \vdash b_1$. Because "iff" is transitive we obtain $\Delta; \Gamma \vdash b_0$ if and only if $\Delta; \Gamma \vdash b_1$, as required.

- case $b = b' \implies C \circ b = C \circ b'$ (congruence).

  Read the statement as a rule: $\dfrac{b = b'}{C \circ b = C \circ b'}$.

Assume: $C \circ b = C \circ b'$. Show: $\Delta; \Gamma \vdash C \circ b$ if and only if $\Delta; \Gamma \vdash C \circ b'$. We must have had $b = b'$. By induction hypothesis on $b = b'$ we obtain $\Delta; \Theta \vdash b$ if and only if $\Gamma; \Theta \vdash b'$, for some Θ. Type derivations of $C \circ b$ and $C \circ b'$ must have ended with the composition rule. Thus, we just need to argue that we have a derivation of $\Theta; \Gamma \vdash C$ if and only if $\Theta; \Gamma \vdash C$, but this is immediate (because "iff" is reflexive).

□

Corollary 10.23 (Decompositionality)**.** *If* $\Delta; \Gamma \vdash b$ *and* $b = b_1 \circ b_0$ *then there exists a typing* Θ *such that* $\Delta; \Theta \vdash b_0$ *and* $\Theta; \Gamma \vdash b_1$*.*

Proof. The Main Lemma yiels $\Gamma; \Delta \vdash b_1 \circ b_0$. By Inversion there exists Θ such that $\Delta; \Gamma \vdash b_0$ and $\Theta; \Gamma \vdash b_1$. □

Theorem 10.2 (Subject Reduction)**.** *For process bigraphs* b_0 *and* b_1*, if* $\Gamma_\emptyset; \Delta \vdash b_0$ *and* $b_0 \rightarrow\!\!\!\!\triangleright\, b_1$ *then* $\Gamma_\emptyset; \Delta \vdash b_1$*.*

Proof. The proof is by analysis of the derivation of $b_0 \rightarrow\!\!\!\!\triangleright\, b_1$ by the sole reaction rule. When reading this proof the reader is recommended to look at the proof trees following the subsequent two paragraphs of the proof.

Because $b_0 \rightarrow\!\!\!\!\triangleright\, b_1$, then by Definition 10.1 there exists an active context D such that $b_0 = D \circ r$ and $b_1 \simeq D \circ r'$. Assume a derivation of $\Gamma_\emptyset; \Delta \vdash b_0$, then also $(*)$ $\Gamma_\emptyset; \Delta \vdash D \circ r$ by Lemma 10.12. By Inversion we must have (among others) the following six subderivations from $(*)$: $(1')$ $\Gamma_\emptyset; \Gamma, y : S \vdash d$, $(3')$ $a : U \vdash a : \mathsf{o}T$, $(3'')$ $x : U' \vdash x : T$, $(4')$ $a : R \vdash a : \mathsf{i}S$, $(5')$ $U \leq R$, and $(8')$ $\Gamma', a : W, x : W'; \Delta \vdash D$. We also know that $W \leq U$ and $W' \leq U'$. By $(3')$, $(5')$ and $(4')$ we conclude $T \leq S$ (cf. [SW01]). $W' \leq U'$, and by $(3'')$ we have $U' \leq T \leq S$, so $W' \leq S$.

Now, consider the derivation to be built. $b_1 \simeq D \circ r'$ implies that $b_1 = D \circ r'$ abstractly. By Lemma 10.12 it suffices to derive $\Gamma_\emptyset; \Delta \vdash D \circ r'$. Reuse the derivation of D. $\varrho = \mathsf{id}_2$ so $\varrho(d) = d$. This means that we can also reuse $(1')$. We still need to justify a derivation of $y : S; x : W' \vdash {}^{(x)}/_{(y)}$. This merely requires justification of $W' \leq S$ because we may choose not to subtype in the other substitutions. $W' \leq S$ has already been established so we can build the desired derivation of $\Gamma_\emptyset; \Delta \vdash D \circ r'$.

Let $A \triangleq B$ signify that A is defined as B.

Left-hand side : Recall that $r = (\mathsf{id}_X \otimes R) \circ d$. Define:

- $r \triangleq \alpha \circ d$

- $\sigma \triangleq {}^{a}/_{a,a'} \otimes {}^{x}/_{x}$
- $\tau \triangleq {}^{a'}/_{a}$
- $\alpha \triangleq \mathsf{id}_X \otimes ((\mathit{join} \otimes \mathsf{id}_{(ax)}) \circ (\sigma \circ (\mathsf{send}_{ax} \otimes (\tau \circ \mathsf{get}_{a(y)}))))$
- $\beta \triangleq (\mathit{join} \otimes \mathsf{id}_{(ax)}) \circ (\sigma \circ (\mathsf{send}_{ax} \otimes (\tau \circ \mathsf{get}_{a(y)})))$
- $\gamma \triangleq \sigma \circ (\mathsf{send}_{ax} \otimes (\tau \circ \mathsf{get}_{a(y)}))$
- $\delta \triangleq \mathsf{send}_{ax} \otimes (\tau \circ \mathsf{get}_{a(y)})$
- $\epsilon \triangleq \tau \circ \mathsf{get}_{a(y)}$

$$\dfrac{\dfrac{(1)\quad \dfrac{(2)\quad \dfrac{\dfrac{\dfrac{(4)\quad (5)}{y : S; a : U' \vdash \epsilon}\ (3)}{y : S; a' : U, a : U, x : U' \vdash \delta}\ (6)}{y : S; a : V, x : V' \vdash \gamma}\ (7)}{y : S; a : W, x : W' \vdash \beta}}{\Gamma, y : S; \Gamma', a : W, x : W' \vdash \alpha}}{\Gamma_\emptyset; \Gamma', a : W, x : W' \vdash r}\ (8)}{\Gamma_\emptyset; \Delta \vdash D \circ r}$$

With the following subderivations (1)-(8).

$$(1) : \dfrac{(1')}{\Gamma_\emptyset; \Gamma, y : S \vdash d} \qquad (2) : \dfrac{\dfrac{(2')}{\Gamma'(X) \leq \Gamma(X)}}{\Gamma; \Gamma' \vdash \mathsf{id}_X}$$

Remark: $\Gamma'(X) \leq \Gamma(X)$ holds whenever $\Gamma'(x_i) \leq \Gamma(x_i)$ for all $x_i \in X$.

$$(3) : \dfrac{\dfrac{(3')}{a : U \vdash a : \mathsf{o}T} \qquad \dfrac{(3'')}{x : U' \vdash x : T}}{\Gamma_\emptyset; a : U, x : U' \vdash \mathsf{send}_{ax}} \qquad (4) : \dfrac{\dfrac{(4')}{a : R \vdash a : \mathsf{i}S}}{y : S; a : R \vdash \mathsf{get}_{a(y)}}$$

$$(5) : \dfrac{\dfrac{(5')}{U \leq R}}{a : R; a' : U \vdash \tau}$$

$$(6):\ \dfrac{\dfrac{\dfrac{(6')}{V \leq U}}{a:U, a':U; a:V \vdash {}^{a}/_{a,a'}} \quad \dfrac{\dfrac{(6'')}{V' \leq U'}}{x:U'; x:V' \vdash {}^{x}/_{x}}}{a':U, a:U, x:U'; a:V, x:V' \vdash \sigma}$$

$$(7):\ \dfrac{(7')}{a:V, x:V'; a:W, x:W' \vdash \mathit{join} \otimes \mathsf{id}_{(ax)}} \qquad (8):\ \dfrac{(8')}{\Gamma', a:W, x:W'; \Delta \vdash D}$$

Remark: (7′) is a straightforward subderivation where subtyping of $W \leq V$ and $W' \leq V'$ may occur.

Right-hand side : Recall that $r' = (\mathsf{id}_X \otimes R') \circ d'$, where $d' = \varrho(d)$. Define:

$$\phi \triangleq \mathit{join} \otimes \mathsf{id}_{(ax)} \qquad \psi \triangleq \mathit{join} \otimes \mathsf{id}_{(x)} \qquad \Theta \triangleq a:W, x:W' \ .$$

Recall that $R' \stackrel{\text{def}}{=} \phi \circ ((\psi \circ (\mathsf{id}_1 \otimes {}^{(x)}/_{(y)})) \otimes (a))$.

$$\dfrac{\text{(i)} \quad \dfrac{\Gamma, y:S; \Gamma', a:W, x:W' \vdash \mathsf{id}_X \otimes R' \qquad \text{(2)} \quad \dfrac{y:S; \Theta \vdash \phi \circ ((\psi \circ (\mathsf{id}_1 \otimes {}^{(x)}/_{(y)})) \otimes (a))}{y:S; \Theta \vdash (\psi \circ (\mathsf{id}_1 \otimes {}^{(x)}/_{(y)})) \otimes (a)}\text{(vi)}}{\Gamma, y:S; \Gamma', a:W, x:W' \vdash r'}\text{(8)}}{\Gamma_\emptyset; \Gamma', a:W, x:W' \vdash r'} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{(iii)}}{y:S; x:W' \vdash {}^{(x)}/_{(y)}}}{y:S; x:W' \vdash \mathsf{id}_1 \otimes {}^{(x)}/_{(y)}}\text{(iv)}}{y:S; x:W' \vdash \psi \circ (\mathsf{id}_1 \otimes {}^{(x)}/_{(y)})}\text{(v)}}{y:S; \Theta \vdash (\psi \circ (\mathsf{id}_1 \otimes {}^{(x)}/_{(y)})) \otimes (a)}\text{(vi)}}{y:S; \Theta \vdash \phi \circ ((\psi \circ (\mathsf{id}_1 \otimes {}^{(x)}/_{(y)})) \otimes (a))}\text{(2)}$$

$$\dfrac{\text{(i)} \quad \dfrac{(1')}{\Gamma_\emptyset; \Gamma, y:S \vdash \varrho(d)} \qquad \text{(ii)} \quad \dfrac{}{\Gamma_\emptyset; \Gamma_\emptyset \vdash \mathsf{id}_1}}{\Gamma_\emptyset; \Gamma, y:S \vdash d} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(iii')}{W' \leq S}}{y:S; x:W' \vdash {}^{x}/_{y}} \quad \dfrac{}{\Gamma_\emptyset; \Gamma_\emptyset \vdash \mathsf{id}_1}}{y:S; x:W' \vdash {}^{x}/_{y} \otimes \mathsf{id}_1} \quad \dfrac{}{y:S; y:S \vdash \ulcorner y \urcorner}}{y:S; x:W' \vdash ({}^{x}/_{y} \otimes \mathsf{id}_1) \circ \ulcorner y \urcorner}$$

With the following subderivations (i)-(vi).

Remark: In (iii), ${}^{(x)}/_{(y)} \stackrel{\text{def}}{=} (x)(({}^{x}/_{y} \otimes \mathsf{id}_1) \circ \ulcorner y \urcorner)$.

$$(\text{iv}) : \frac{(\text{iv}')}{x : W'; x : W' \vdash \psi} \qquad (\text{v}) : \frac{\overline{\Gamma_\emptyset; a : W \vdash {}^{a}/_{\emptyset}}}{\Gamma_\emptyset; a : W \vdash (a)({}^{a}/_{\emptyset})}$$

Remarks: In the subderivation (iv′) we choose not to subtype. In derivation (v), $(a) \stackrel{\text{def}}{=} (a)({}^{a}/_{\emptyset})$, and any type may be chosen for a – pick W.

$$(\text{vi}) : \frac{(\text{vi}')}{a : W, x : W'; a : W, x : W' \vdash \phi}$$

Remark: In the subderivation (vi′) we choose not to subtype. □

Proposition 10.3 (Type Soundness)**.** *Suppose that process bigraph $b = [\![P]\!]_{(X)}$, $\Gamma_\emptyset; \Gamma \vdash b$, and $b \rightarrow^* b'$. Then, for each non-idle $a \in glob(cod(b'))$ it holds that:*

1. *If $\Gamma \vdash a : \mathsf{i}S$ then a is either linked to the channel port of a* get *ion or linked to the datum port of a* send *ion.*
2. *If $\Gamma \vdash a : \mathsf{o}T$ then a is linked to a* send *ion.*

Proof. The proof is by induction on the length n of the reduction $b \rightarrow^* b'$. Notice that for process bigraphs non-idle outer names must have preimages under the link map that are ports.

- $n = 0$. We have $b = b'$. Either $\Gamma(a) = \mathsf{i}S$ or $\Gamma(a) = \mathsf{o}T$.

  Assume $\Gamma(a) = \mathsf{i}S$. Clearly, a can be linked to the channel port of a get ion with type $\mathsf{i}S'$ for $S \leq S'$ (and thus $\mathsf{i}S \leq \mathsf{i}S'$), because the rule for typing substitutions ${}^{y}/_{X}$ enforces $\Gamma(y) = \Gamma(X)$. (Subtyping can happen by identities, for instance.) Name a can *not* be linked to the channel port of a get ion, because that would require a to have a type of form $\mathsf{o}T$ (for some T) by the typing rule for send, which contradicts the assumption. Nevetherless, a can be linked to the datum port of a send control, say send_{ca}, because links a of input type can be communicated, as long as $\Gamma \vdash c : \mathsf{o}\mathsf{i}S$ and $\Gamma' \vdash a : \mathsf{i}S$ for some Γ, Γ', S, and T.

  Assume $\Gamma(a) = \mathsf{o}T$. Clearly a must be linked to a send ion because the premise of the get rule requires a to have a type of form $\mathsf{i}S$, which contradicts the assumption. Name a can either be linked to a channel port of send or the datum port if it is to be communicated.

- $n > 0$. Assume the induction hypothesis for $b \rightarrow\!\triangleright^n b^n$ and show it for $b \rightarrow\!\triangleright^{n+1} b'$. Thus, b^n exhibits the property. We show that b^{n+1} does too. By Subject Reduction we know that $\Gamma_\emptyset; \Gamma \vdash b^n$ and taking another step with $\rightarrow\!\triangleright^1$ establishes $\Gamma_\emptyset; \Gamma \vdash b^{n+1}$. The outer typing being preserved we just need arguments as for $n = 0$ to establish the *desiderata*.

□

Lemma 10.4 (Weakening)**.** *If* $\Delta; \Gamma \vdash b$ *and* $x \notin \mathsf{supp}(\Gamma)$ *then* $\Delta; \Gamma, x : T \vdash b \otimes (x)$.

Proof. The proof is by induction on the height of the derivation of $\Delta; \Gamma \vdash b$. Recall that $(x) \stackrel{\text{def}}{=} (x)(x/\emptyset)$ and can thus be typed by using the rule for abstraction and then the rule for substitutions.

All cases follow the same pattern, where the desired derivation is constructed as follows:

$$\dfrac{\dfrac{\text{"by assumption"}}{\Delta; \Gamma \vdash b} \qquad \dfrac{\dfrac{}{\Gamma_\emptyset; x : T \vdash x/\emptyset}}{\Gamma_\emptyset; x : T \vdash (x)}}{\Delta; \Gamma \vdash b \otimes (x)}$$

The case for closure $/x$ holds by the insight that an outer name x is different from an inner name x if they are not linked. □

Lemma 10.5 (Strengthening)**.** *If* $\Delta; \Gamma, x : T \vdash b \otimes (x)$ *then* $\Delta; \Gamma \vdash b$.

Proof. The proof is by induction on the height of the derivation of $\Delta; \Gamma, x : T \vdash b \otimes (x)$. Recall that $(x) \stackrel{\text{def}}{=} (x)(x/\emptyset)$ and can thus be typed by using the rule for abstraction and then the rule for substitutions.

All cases follow the same pattern, where the assumed derivation has form:

$$\dfrac{\dfrac{(*)}{\Delta; \Gamma \vdash b} \qquad \Gamma_\emptyset; x : T \vdash (x)}{\Delta; \Gamma, x : T \vdash b \otimes (x)}$$

The desired derivation is simply constructed by reusing (*). □

Lemma 10.6. *Suppose* $b = [\![P]\!]_{(X)}$ *and* $\Gamma_\emptyset; \Gamma \vdash b$ *with* $\mathsf{fn}(P) \subseteq X = \mathsf{supp}(\Gamma)$. *Then* $[\![P]\!]_{\mathsf{supp}(\Gamma)} \otimes (x) = [\![P]\!]_{(\mathsf{supp}(\Gamma, x:T))}$ *for any* T.

Proof. The proof is by structural induction on P. Let LHS mean "left-hand side" and RHS "right-hand side". We sometimes write Xy for $X \uplus \{y\}$, for instance.

- case $P = \mathbf{0}$.

  LHS: $[\![\mathbf{0}]\!]_{\mathsf{supp}(\Gamma)} \otimes (x) \stackrel{\text{def}}{=} (X) \otimes (x) = (X \uplus \{x\})$. RHS: $[\![\mathbf{0}]\!]_{\mathsf{supp}(\Gamma, x:T)} = (X \uplus \{x\})$.

- case $P = \nu z\, Q$.

  LHS: $[\![\nu z\, Q]\!]_{(X)} \otimes (x) \stackrel{\text{def}}{=} /(z) \lhd \mathsf{id}_X \circ [\![Q]\!]_{(Xz)} \otimes (x)$.

  RHS: $[\![\nu z\, Q]\!]_{(\mathsf{supp}(\Gamma, x:T))} \stackrel{\text{def}}{=} /(z) \lhd \mathsf{id}_{Xx} \circ [\![Q]\!]_{(Xxz)}$. We know that x is idle in $[\![Q]\!]$ so we may further rewrite to $/(z) \lhd \mathsf{id}_X \circ [\![Q]\!]_{Xz} \otimes (x)$.

- case $P = \overline{a}z.Q$.

  LHS: $[\![\overline{a}z.Q]\!]_{(X)} \otimes (x) \stackrel{\text{def}}{=} \mathsf{send}_{az} \lhd \mathsf{id}_X \circ [\![Q]\!]_{(X)} \otimes (x)$.

  RHS: $[\![\overline{a}z.Q]\!]_{(\mathsf{supp}(\Gamma, x:T))} \stackrel{\text{def}}{=} \mathsf{send}_{az} \lhd \mathsf{id}_{Xx} \circ [\![Q]\!]_{(Xx)} = \mathsf{send}_{az} \lhd \mathsf{id}_{Xx} \circ [\![Q]\!]_{(X)} \otimes (x)$ because x is idle in $[\![Q]\!]$.

- case $P = a(y).Q$.

  LHS: $[\![a(y).Q]\!]_{(X)} \otimes (x) \stackrel{\text{def}}{=} \mathsf{get}_{a(y)} \lhd \mathsf{id}_X \circ [\![Q]\!]_{(X)} \otimes (x)$.

  RHS: $[\![a(y).Q]\!]_{(\mathsf{supp}(\Gamma, x:T))} \stackrel{\text{def}}{=} \mathsf{get}_{a(y)} \lhd \mathsf{id}_{Xx} \circ [\![Q]\!]_{(Xx)} \otimes (x) = \mathsf{get}_{a(y)} \lhd \mathsf{id}_X \circ [\![Q]\!]_{(X)} \otimes (x)$ because x idle in $[\![Q]\!]$.

- case $P = Q \mid Q'$.

  LHS: $[\![Q \mid Q']\!]_{(X)} \otimes (x) \stackrel{\text{def}}{=} ([\![Q]\!]_{(X)} \mid [\![Q']\!]_{(X)}) \otimes (x)$.

  RHS: $[\![Q \mid Q']\!]_{(\mathsf{supp}(\Gamma, x:T))} \stackrel{\text{def}}{=} [\![Q]\!]_{(\mathsf{supp}(\Gamma, x:T))} \mid [\![Q']\!]_{(\mathsf{supp}(\Gamma, x:T))} \stackrel{\text{def}}{=} [\![Q]\!]_{(Xx)} \mid [\![Q']\!]_{(Xx)} = [\![Q]\!]_{(X)} \mid [\![Q']\!]_{(X)} \otimes (x)$ because x is idle in $[\![Q]\!]$ and $[\![Q']\!]$.

□

Proposition 10.7 (Transfer of Type Derivations)**.** $\Gamma \vdash P : \diamond$ *if and only if* $\Gamma_\emptyset; \Gamma \vdash [\![P]\!]_{(X)}$ *when* $\mathsf{fn}(P) \subseteq X = \mathsf{supp}(\Gamma)$.

Proof. The proof is by struct. induction on P using Lemmas 10.18 and 10.12.

- case $P = \mathbf{0}$.

  "⇒": Assume $\Gamma \vdash \mathbf{0} : \diamond$. Recall: $(X) \stackrel{\text{def}}{=} (X)(^X/_\emptyset) \stackrel{\text{def}}{=} (X)(\bigotimes_{i=1}^n x_i\emptyset)$. So, we need to establish $\Gamma \restriction x_i \vdash {}^{x_i}/_\emptyset$ (for $i = 1..n$), but they are all trivial by the rule for substitution, and then use the rule for abstraction to build $\Gamma_\emptyset; \Gamma \vdash (X)$, as required.

  "⇐": Assume $\Gamma_\emptyset; \Gamma \vdash (X)$. Build $\Gamma \vdash \mathbf{0} : \diamond$ by axiom.

- case $P = (\nu x : L)\, Q$.

  "$\Rightarrow$": Assume $\Gamma \vdash (\nu x : L)\, Q : \diamond$ with premise (*) $\Gamma, x : L \vdash Q$. Observe that $[\![(\nu x : L)\, Q]\!]_{(X)} \stackrel{\text{def}}{=} /(x) \lhd \mathsf{id}_X \circ [\![Q]\!]_{(Xx)} = (X)\Big((/x \otimes \mathsf{id}_X) \circ \ulcorner Xx \urcorner \circ [\![Q]\!]_{(Xx)}\Big)$. By the Main Lemma it is enough to show a derivation for this last term. We need to establish four premises: (1) $\Gamma_\emptyset; \Gamma, x : L \vdash [\![Q]\!]_{(Xx)}$, (2) $\Gamma, x : L; \Gamma, x : L \vdash \ulcorner Xx \urcorner$, (3) $x : L; \Gamma_\emptyset \vdash /x$, and (4) $\Gamma; \Gamma \vdash \mathsf{id}_X$. (1) follows by induction hypothesis on (*) and (2)-(4) from axioms.

  "$\Leftarrow$": Simply by induction hypothesis.

- case $P = \bar{a}x.Q$.

  "$\Rightarrow$": Assume $\Gamma \vdash \bar{a}x.Q : \diamond$ with premises (1) $\Gamma \vdash a : \mathsf{o}T$, (2) $\Gamma \vdash x : T$, and (3) $\Gamma \vdash Q : \diamond$. Observe that $[\![\bar{a}x.Q]\!]_{(X)} \stackrel{\text{def}}{=} \mathsf{send}_{ax} \lhd \mathsf{id}_X \circ [\![Q]\!]_{(X)} = (X)(\mathsf{send}_{ax} \otimes \mathsf{id}_X \otimes \mathsf{id}_1) \circ [\![Q]\!]_{(X)} = (X)(\mathsf{send}_{ax} \otimes \mathsf{id}_X \otimes \mathsf{id}_1 \circ \ulcorner X \urcorner \circ [\![Q]\!]_{(X)})$. By the Main Lemma it is enough to show a derivation for this last term. We need to establish three premises: (i) $\Gamma_\emptyset; \Gamma \vdash [\![Q]\!]_{(X)}$, (ii) $\Gamma \restriction \{a\} \vdash a : \mathsf{o}T$, and (iii) $\Gamma \restriction \{x\} \vdash x : T$. (i) is by induction hypothesis on (3), (ii) is follows from (1), and (iii) follows from (2).

  "$\Leftarrow$": Reverse the argument on premises.

- case $P = a(y).Q$.

  "$\Rightarrow$": Assume $\Gamma \vdash a(y).Q : \diamond$ with premises (1) $\Gamma \vdash a : \mathsf{i}S$ and (2) $\Gamma, y : S \vdash Q : \diamond$. We have $[\![a(y).Q]\!]_{(X)} \stackrel{\text{def}}{=} \mathsf{get}_{a(y)} \lhd \mathsf{id}_X \circ [\![Q]\!]_{(Xy)}$. We need to establish (i) and (ii) $\Gamma_\emptyset; \Gamma, y : S \vdash [\![Q]\!]_{(Xy)}$ to build $\Gamma_\emptyset; \Gamma \vdash \mathsf{get}_{a(y)} \lhd \mathsf{id}_X \circ [\![Q]\!]_{(Xy)}$. (i) follows from (1) and (ii) follows by induction hypothesis on (2).

  "$\Leftarrow$": By induction hypothesis and the fact that (1) follows from (i).

- case $P = Q \mid Q'$.

  "$\Rightarrow$": Assume $\Gamma \vdash Q \mid Q' : \diamond$ with premises (1) $\Gamma \vdash Q : \diamond$ and (2) $\Gamma \vdash Q' : \diamond$. We have $[\![Q \mid Q']\!]_{(X)} \stackrel{\text{def}}{=} \sigma \circ ([\![Q]\!]_{(X)} \otimes \tau \circ [\![Q']\!]_{(X)})$, for suitable substitutions σ and τ.

  Now, let $W = \mathsf{fn}(Q) \cap \mathsf{fn}(Q')$. Then, if $Z = \mathsf{fn}(Q) \setminus W$ then $\tau = {}^{Z}/_{Z} \otimes {}^{F}/_{W}$ and $Y = Z \uplus F$. To build $\Gamma_\emptyset; \Gamma \vdash \sigma \circ ([\![Q]\!]_{(X)} \otimes \tau \circ [\![Q']\!]_{(X)})$ we must establish three premises: (i) $\Gamma_\emptyset; \Gamma \restriction Q \vdash [\![Q]\!]_{(\mathsf{fn}(Q))}$, (ii) $\Gamma_\emptyset; \Gamma \restriction Q' \vdash [\![Q']\!]_{(\mathsf{fn}(Q'))}$, (iii) $\Gamma \restriction Q'; \Gamma \restriction Y \vdash \tau$, and (iv) $(\Gamma \restriction Q), (\Gamma \restriction Y); \Gamma \vdash \sigma$. By induction hypothesis on (1) and (2) we obtain $\Gamma_\emptyset; \Gamma \vdash [\![Q]\!]_{(X)}$ and $\Gamma_\emptyset; \Gamma \vdash [\![Q']\!]_{(X)}$. Then, by Lemma 10.18 and Strengthening we obtain $\Gamma_\emptyset; \Gamma \restriction \mathsf{fn}(Q) \vdash [\![Q]\!]_{(X)}$ and

$\Gamma_\emptyset; \Gamma \restriction \mathsf{fn}(Q') \vdash [\![Q']\!]_{(X)}$, because $X \setminus \mathsf{fn}(Q)$ is idle in $[\![Q]\!]$ and likewise for Q'. (iii) and (iv) follow trivially from the rule for substitutions, where we choose not to subtype.

"$\Leftarrow$": Assume $\Gamma_\emptyset; \Gamma \vdash \sigma \circ ([\![Q]\!]_{(X)} \otimes \tau \circ [\![Q']\!]_{(X)})$ with the following four premises (i) $\Gamma_\emptyset; \Gamma \restriction \mathsf{fn}(Q) \vdash [\![Q]\!]_{(\mathsf{fn}(Q))}$, (ii) $\Gamma_\emptyset; \Gamma \restriction \mathsf{fn}(Q') \vdash [\![Q']\!]_{(\mathsf{fn}(Q'))}$, (iii) $\Gamma \restriction \mathsf{fn}(Q'); \Gamma \restriction Y \vdash \tau$, and (iv) $(\Gamma \restriction \mathsf{fn}(Q)), (\Gamma \restriction Y); \Gamma \vdash \sigma$. (i) and (ii) imply (1) and (2), respectively, by the induction hypothesis, Lemma 10.18, and Weakening. τ and σ are immaterial because when using Weakening we pick the "right" types. Thus, we have established $\Gamma \vdash Q : \diamond$ and $\Gamma \vdash Q' : \diamond$ so by the rule for parallel composition we obtain $\Gamma \vdash Q \mid Q' : \diamond$, as required.

□

11

On Type Systems and Sortings for Bigraphs

In this chapter we study the relationship between sortings and inductive type systems for process calculi at the meta-level of Bigraphs. We work toward a notion that we call *inductive sortings*. It is work in progress, jointly with Mikkel Bundgaard, Søren Debois, and Thomas Hildebrandt.

In Chapter 10 we saw how one can define type systems for Bigraphs inductively on the structure of the elementary bigraphs. The approach is to define them in the classical way as known from type systems for λ-calculus and for process calculi, for instance. One advantage of this approach is that well-known type systems can be recovered and new ones discovered in the classical format. In our work on inductive type systems for Bigraphs we barely consider LTSs and bisimulation congruences. In fact, because the inductive type systems are defined on a BRSs without altering the underlying category the categorical machinery of Bigraphs remains intact.

Nevertheless, there exists another theory of types or sorts for bigraphs. Namely that of *sortings* (or sorting functors) by Debois and co-workers [BDH08, Deb08], which takes further the work on place and link sortings by Milner, Leifer, and Høgh Jensen [LM06, Jen07, JM04]. It is the work on sorting functors (a semantic approach), which we aim to bridge with our work on inductive type systems (a syntactic approach).

11.1 Introduction to sortings

Recall that BRSs are built on top of s-categories (see Appendix A.1). Often, we may have a category $\mathcal{C}$ with RPOs, but when a BRS is constructed on top of it we obtain a behavioural equivalence (on bigraphs) that is awry because the automatically derived labels do not correspond to the observable behaviour we wish to model. To solve this we may derive a new category $\mathcal{S}$, which resembles $\mathcal{C}$, by enriching the objects and then omitting morphisms

that do not satisfy a certain (structural) condition of our choosing. This procedure is known as "constructing a sorting". Then, we build a BRS on $\mathcal{S}$, which by construction will have the desired behavioural equivalence. $\mathcal{S}$ is a "sorting" of $\mathcal{C}$.

11.1.1 Sortings as functors

Milner, Leifer, and Høgh Jensen realised that stipulating a condition or predicate on morphisms to weed out unwanted ones, gives rise to a forgetful functor F from a sorted category $\mathcal{S}$ into the original category $\mathcal{C}$. Debois, Birkedal, and Hildebrandt suggested instead to take the functor F as the definition of a sorting, because this allowed them to use standard category theory in developing sorting as a mathematical tool.

Definition 11.1 (Sorting, [Deb08])**.** *A* sorting *of a category $\mathcal{C}$ is a functor $F : \mathcal{S} \to \mathcal{C}$ that is faithful and surjective on objects.*

We are not merely interested in the category underlying a BRS, but in the BRS and its equivalences as such. So, how does such a sorting help us refine the category underlying the BRS? Essentially, a sorting enriches the interfaces (objects) to obtain finer control of bigraph (morphism) composition. The interfaces are enriched by adding another component, namely a sort. So, for a bigraph $F(f) : \langle n, loc, X\rangle \to \langle m, loc', Y\rangle$ in $\mathcal{C}$ we have $f : \langle n, loc, X, sort\rangle \to \langle m, loc', Y, sort'\rangle$ in the sorted category $\mathcal{S}$. The same object can be enriched in many different ways corresponding to giving a term different types. We can refine a homset in $\mathcal{C}$ into several homsets in $\mathcal{S}$; if $\mathcal{C}$ has a homset $\mathcal{C}(a, b)$, $\mathcal{S}$ will have a homset $\mathcal{C}(a', b')$ for every pair of enrichments a' of a and b' of b. The homset is split up between the refined objects in such a way that each such homset in $\mathcal{S}$ will contain a subset of the morphisms of the original homset in $\mathcal{C}$. Theorem 3.45 of [Deb08] provides sufficient conditions for a sorting approximating a predicate P to admit the same BRSs as $\mathcal{C}$ did up to P, in a sense preserving reaction and transition semantics. We will not go into that here, but to mention that P has to be decomposable, i.e., $P(f \circ g)$ implies $P(f)$ and $P(g)$. In other words, whenever the predicate holds for a morphism, it also holds for the constituents of any (vertical) decomposition of that morphism.

11.1.2 Predicate sortings

From [Deb08] we know how to construct a sorting $F : \mathcal{S} \to \mathcal{C}$ for a given decomposable predicate P, which 1) transfers RPOs from $\mathcal{C}$ to $\mathcal{S}$ so that

bisimilarity will be a congruence, and 2) preserves semantics of the BRS to the extent allowed by P. It is noted in [Deb08] that for free structures, imposing a decomposable predicate corresponds to ruling out terms containing particular sub-terms. Such predicates are usually given in plain English, albeit informally translatable to first-order logic formulae. The theory of sortings is parametrised on the language used for defining the predicates. More importantly, these predicates are often non-inductive. An example of a predicate is that "no nodes with control A may be nested immediately inside nodes with control B." Predicates are often structural in this way, see the survey in Chapter 6 of [Deb08]. Traditionally, predicates state something about the structure of morphisms (bigraphs) rather than their behaviour.

11.1.3 Closure sortings

We will briefly provide the reader with the most basic intuitions about closure sortings, a particular bread of predicate sortings. Closure sortings are the most advanced and useful generic sortings. Our errand is not to recapitulate the theory in detail. For that see [Deb08]. Closure sortings transfer RPOs and have semantic correspondence. (See [Deb08] for a formal definition of "have semantic correspondence".) These are the key properties required of a sorting to be useful with respect to BRSs and the above-mentioned problem with labels. The following two paragraphs recapitulate some essential intuitions of [Deb08].

In the sorted category $\mathcal{S}$, the objects become more complicated than in the original category $\mathcal{C}$. According to [Deb08], given morphisms $f : a \to b$ and $g : b \to c$ in $\mathcal{C}$, when constructing a sorting, "we must decide, for every pre-image b' of b, which of f and g should have a pre-image with codomain b' respectively domain b'. Obviously, it cannot be both." The closure sorting expresses this choice in each object of $\mathcal{C}$.

The objects of $\mathcal{S}$ are on form $(U, V)_b$, where $b \in Ob(\mathcal{C})$, U is a set of morphisms with codomain b, and V is a set of morphisms with domain b. The sorting requires that composition of any morphism in U with a morphism in V must yield a morphism satisfying predicate P, written $U \perp V$[1]. We say that F is *prefix-closed* if $f \circ g \in F$ implies $g \in F$, and that F is *suffix-closed* if $f \circ g \in F$ implies $f \in F$.

Roughly, there is $f : (U, V)_b \to (W, T)_c \in Mo(\mathcal{S})$ when

- $f : b \to c \in Mo(\mathcal{C})$,

[1] Not to be confused with the notation of Chapter 3.

- f satisfies P,
- both $f \in V$ and $f \in W$,
- every object $(F, G)_c$ has F suffix-closed and G prefix-closed,
- and every object is maximal in the sense that it is impossible to add more morphisms to either F or G without violating the condition $G \perp F$.

Thus, a closure sorting records in each object a of $\mathcal{S}$ which morphisms of $\mathcal{C}$ are admitted at a.

11.2 Inductive sortings

In this section we will discuss how an inductive type system on a BRS can be seen as a decomposable predicate on morphisms. This is a first step in bridging the work of Chapter 10 with the work on sorting functors. To bridge the two branches of research we explore to which extent decomposable predicates can be given inductively, and more importantly, to which extent they can capture traditional type systems for process calculi. Usually, a traditional type system will enjoy a type soundness theorem stating that well-typed terms behave well, in some precise sense. In Chapter 10, type soundness stated that public names are used in accordance with the i/o typing discipline. Sortings generated from decomposable predicates aim to remove unwanted behaviour of the BRS by removing some contexts, thus trimming the set of labels in the derived LTS. On the other hand, in Chapter 10 we used a bigraphical inductive type system to outlaw ill-behaved bigraphs.

11.2.1 Type systems as predicates

An inductive type system T over a BRS can be seen as a predicate P on bigraph terms as follows. If we can show a result like the Main Lemma of Chapter 10, i.e., if $\Delta; \Gamma \vdash b$ and $b = b'$, then $\Delta; \Gamma \vdash b'$, we will have typing on the graphs underlying the term representation (because the term language is sound and complete). Hence, the bigraphs for which P holds are the well-typed ones. Notice that we have not introduced any enrichment of objects or refinement of homsets, yet. Since T is inductively defined, P will necessarily become decomposable, at least if the typing rule for categorical composition is as the natural one of Chapter 10: if $\Delta; \Theta \vdash b$ and $\Theta; \Gamma \vdash b'$, then $\Delta; \Gamma \vdash b' \circ b$. Whence, we can construct a closure sorting $\mathcal{S}_P : \mathcal{S} \to \mathcal{C}$

over P, from a sorted category $\mathcal{S}$ into the original category C. We do so by stipulating, for $f \in Mo(\mathcal{S})$, that

$$P(f) \text{ iff there exist } \Delta \text{ and } \Gamma \text{ such that } \Delta;\Gamma \vdash f_{rep} \ ,$$

where f_{rep} is a term representation (up to α-equivalence on binders) of f; f_{rep} is a representative from the equivalence class of terms that correspond to f. We know that the homsets of C are split up in $\mathcal{S}$, but the morphisms in the homsets remain unchanged. The objects of $\mathcal{S}$, however, will be quite different and most likely larger in number.

It is an open question how the objects in $\mathcal{S}$ will look, for instance in the case of the i/o type system. Objects in $\mathcal{S}$ have an extra component, the sort, in comparison with objects in C. Furthermore, what we are really interested in, is to know how the typings on morphisms in C relate to objects in $\mathcal{S}$. We want to know this because the typings can be read and understood easily, and we would rather not lose that property when we move to sortings and consider the objects in $\mathcal{S}$. Sortings are faithful functors so they relate homsets. Recall that given a derivation $\Delta;\Gamma \vdash b$, we have $\mathsf{supp}(\Delta) = glob(dom(b))$ and $\mathsf{supp}(\Gamma) = glob(cod(b))$. So, the question is: Given a derivation $\Delta;\Gamma \vdash b$, how does Δ relate to the domain of each f' in the pre-image (a set of morphisms) of f? (And likewise for the codomain of f' and Γ.) We leave that to future work.

Armed with the insight above, and despite the current uncertainty about the relation between sorted objects and typings, we proceed to discuss how we may generate an LTS over typed terms from an untyped calculus.

11.2.2 Generating LTSs over typed terms

Our idea is as follows. Given an untyped calculus K of inductively defined terms equipped with a reaction semantics, and a semantically corresponding BRS $\acute{}\mathrm{B}_C$ on category C, we

1. define a (sound) inductive type system T_C on $\acute{}\mathrm{B}_C$,
2. consider T_C as a decomposable predicate P_{T_C},
3. construct a closure sorting from P_{T_C},
4. erect a new BRS $\acute{}\mathrm{B}_{\mathcal{S}}$, which has semantic correspondence with $\acute{}\mathrm{B}_C$, on the sorted category $\mathcal{S}$,
5. derive an LTS $L_{\mathcal{S}}$ with congruential bisimulation on $\acute{}\mathrm{B}_{\mathcal{S}}$, and

6. transfer $L_{\mathcal{S}}$ to K.

So, what do we achieve? Given 1) a calculus K, 2) a bigraphical model B_C for it, and 3) a structural and dynamic correspondence between K and $\acute{}\mathrm{B}_C$, we can A) define an inductive type T_C system on $\acute{}\mathrm{B}_C$ and then obtain an LTS $L_{\mathcal{S}}$ with congruential bisimulation on well-typed (well-sorted) terms of $\acute{}\mathrm{B}_C$ (namely the morphisms of $\mathcal{S}$), and thus also on well-typed terms of K by the correspondences.

Step 1 requires a little bit of work, but if we can progress further with a generic type system for message-passing and other process calculi, then this task will diminish in size.

Steps 2 through 5 are automatic, and the machinery for steps 3 through 5 was developed in [Deb08]. Step 6 is semi-automatic because the transfer of a bigraphical inductive type system may require some work if the operators of K have complicated images in B_C, as discussed in Chapter 10. Unfortunately, we can only obtain transfer of $L_{\mathcal{S}}$ to K if the following statement holds:

$$P \sim_K Q \text{ iff } [\![P]\!] \sim_{B_C} [\![Q]\!] ,$$

where P, Q are K-processes and $\sim_K$ the derived a behavioural equivalence. Often, this does not hold precisely, but is close to, cf. [Jen07, Deb08]. The behavioural equivalence(s) may be almost the one(s) we intended. An example is Høgh Jensen's model of the full π-calculus [Jen07]. Even in the cases where we can not directly transfer the LTS, we may gain insights by considering the (almost intended) LTS on the bigraphical model.

11.2.3 Type soundness as a predicate

One can even imagine defining type soundness of the inductive bigraphical type system as a decomposable predicate, e.g., when the inductive type system is not known or difficult to obtain or define. This would increase the value of the derived LTS, because then the bigraphs would satisfy a behavioural property instead of a structural one. Unfortunately, it seems difficult to capture type soundness of a non-trivial inductive type system as a *decidable* decomposable predicate. Nevertheless, it may prove feasible and useful to define approximations to type soundness, i.e., a weaker property, as a decidable decomposable predicate. An example could be to define type soundness of polyadic π-calculus, where we are interested in catching arity mismatches in communication. We could try to capture that property with the following predicate on morphisms f of a bigraphical model of polyadic π-calculus:

$$P(f) \text{ iff } Q \twoheadrightarrow^* Q' ,$$

where $\twoheadrightarrow^*$ is the reflexive and transitive closure of $\twoheadrightarrow$, and

$$Q' \stackrel{\text{def}}{=} (\nu z_1, \ldots, z_k)(Q_2 \mid \overline{a}\langle x_1, \ldots, x_n\rangle.Q_3 \mid a(y_1, \ldots, y_m).Q_4) \,,$$

for $m \neq n$ and processes Q_2, Q_3, Q_4. Alas, P is very intentional and clearly undecidable. More work is needed to clarify the potential of this line of research. Perhaps we may find useful insights about static analysis for π-calculi, such as in [BDNN98].

11.3 Summary

We have sketched how a bigraphical inductive type system can be seen as a decomposable predicate. From such a predicate we construct a closure sorting. This closure sorting yields a sorted category. On the sorted category we erect a BRS, which has semantic correspondence with the BRS on the unsorted category – we have merely safely thrown away bigraphs that ruined our labels. Then, we automatically derive an LTS on the sorted BRS. Assuming a bigraphical model of an untyped calculus K, terms given inductively, with a reduction relation, we can, in some cases, transfer the bigraphical type system (and its properties) along with the LTS to a now typed version of K. In the cases where direct transfer is not possible, we may still obtain a better understanding of K by considering its bigraphical model, because it is more abstract.

Future work should make more formal these ideas and apply the approach to a case study, e.g., the i/o typing for π-calculus.

Part IV

Conclusion

12

Summary

And so, we arrive at the conclusion. We have challenged Bigraphs as a theory for ubiquitous computing through experimentation with modelling and simulation of location models of varying complexity. Moreover, we have considered the transition from location-aware to context-aware systems. On top of this we have explored the novel direction of Bigraphs as a meta-model for inductive type systems for process calculi.

Let us summarise our developments:

- We have surveyed the research literature on location models. We have been able to identify symbolic location models as an appropriate case study and synthesise the terminology to distill a representative model for our modelling efforts. (Chapter 2.)

- Realising that direct modelling of a whole location system in Bigraphs causes problems, because we lack control structures when programming directly with bigraphical reaction rules – for instance recursive tree traversals – we have developed Plato-graphical models. These models are useful for modelling and are also interesting from a theoretical point of view, because they show how we may combine several BRSs into one. The modelling effort is acceptable. Nevertheless, some new theoretical challenges arise, an important one being the missing notion of bisimilarity between BRSs. (Chapter 3.)

- The Plato-graphical setup allows us to program or express some parts of our modelled systems in different languages, i.e., a more suitable domain-specific language with more control structures such as MiniML. We exploit this to encode two parts of the location-aware systems in MiniML; the location model and the location-aware application. We have developed the translation from MiniML into Bigraphs,

one of the most interesting technical aspects being the representation of references, which is analysed in detail technically. The translation has also been implemented as an SML match compiler along with some auxiliary programs. (Chapter 4.)

- With the tools of Plato-graphical models and the MiniML encoding we are able to model a realistic location system in Bigraphs. The sentient building case study has helped us gain a better general understanding of which requirements context-aware (location-aware) programming puts on the theory used for the modelling. Furthermore, we have implemented the model in the syntax of the BPL tool. (Chapter 5 and Appendix A.2.)

- To complete the program we have performed simulations of an abstract version of the location system with the BPL tool. The simulations reveal important design decisions and illustrate how intricate it can be to cover all possibilities when working with concurrency and mobility. (Chapter 6.)

- Part II is completed with an extensive survey of related work on formal models for context-awareness. (Chapter 7.)

- The main contribution of Part III is our development of inductive type systems for Bigraphs. The challenge of designing type systems and proving properties about them has been pushed on to the meta-model of Bigraphs. As proof of concept we have presented a bigraphical model of a π-calculus, developed an i/o-type system with subtyping on this model, proved crucial properties (including subject reduction) for this type system, and transferred these properties to the (typed) π-calculus. (Chapter 10.)

- The work on inductive sortings is very preliminary. We sketch how a bigraphical inductive type system can be seen as a decomposable predicate, yielding a closure sorting. The sorted category underlies a sorted BRS for which we automatically derive an LTS. Assuming a bigraphical model of an untyped calculus K we can, under certain circumstances, transfer the bigraphical type system (and its properties) along with the LTS to a now typed version of K. (Chapter 11.)

Summarising, we conclude that Bigraphs is still in play as a theory for modelling and simulating ubiquitous systems. We have taken a step in bringing together the research on location models with the formal theory of

Bigraphs. It is clear that if theoreticians are to truly influence the way that systems are built, then we must try to work with the systems and problems that designers face. Moreover, significant efforts in theory pertaining to Bigraphs (sortings, type systems, Directed Bigraphs, Stochastic Bigraphs and so forth) indicate that Bigraphs is a maturing theory.

13

Future work

We have argued that the techniques used in our work can also encompass context-aware computing. Nevertheless, Bigraphs needs to encompass time and continuous space to become a theory for global ubiquitous computing.

To realise the full potential of Bigraphs more experimentation is needed. We need to further develop the BPL tool to handle models of a larger magnitude. As well, we need to perform more elaborate simulations, and perhaps to enhance the user interface to provide a useful tool for engineers and system designers. (There is a web interface.) The key to success with simulation — in the sense of the discovery of unforeseen system behaviour — is to write elaborate tactics that semi-automatically rewrite the system according to heuristics (or other guide lines). We may even go to model checking, where interfacing with well developed existing tool seems like the way to go. Both simulation and model checking are important utilities if we are to rigorously reason about systems large than toy examples. Another strand of work that is worth attention is the inclusion of sorting or typing into the BPL tool. We agree with Debois [Deb08], who refers to Milner and Leifer, that sorting (or typing) is likely to be needed in any significant application. For a much more detailed account of future work within modelling and simulation, see Chapter 8.)

As mentioned in the future work section of Chapter 10, there is also plenty of interesting work to be done within inductive type systems for process calculi using Bigraphs as a meta-model. Going to more advanced type systems such as deadlock-freedom seems like the most promising direction. Furthermore, the relationship with sortings merits further investigations, but we are quite excited about the prospects.

And so concludes my dissertation.

Appendix

A.1 Background Bigraph definitions

This appendix contains the relevant definitions of [JM04, Mil06a, Jen07].

Definition A.1 (pure signature)**.** *A* (pure) signature $\mathcal{K}$ *is a set whose elements are called* controls*. For each control K it provides a finite ordinal ar*(K)*, an* arity*; it also determines which controls are* atomic*, and which of the non-atomic controls are* active*. Controls which are not active (including the atomic controls) are called* passive*.*

Presuppose a countably infinite set χ of global names.

Definition A.2 (concrete pure bigraph)**.** *A* (concrete) pure bigraph *over the signature* $\mathcal{K}$ *takes the form* $G = (V, E, ctrl, G^{\mathsf{P}}, G^{\mathsf{L}}) : I \to J$ *where* $I = \langle m, X \rangle$ *and* $J = \langle n, Y \rangle$ *are its* inner *and* outer faces*, each combining a* width *(a finite ordinal) with a finite set of global names drawn from* χ*. Its first two components V and E are finite sets of* nodes *and* edges *respectively. The third component* $ctrl : V \to \mathcal{K}$*, a* control map*, assigns a control to each node. The remaining two are:* $G^P = (V, ctrl, prnt) : m \to n$*,* $G^L = (V, E, ctrl, link) : X \to Y$*.*

Definition A.3 (prime interface)**.** *An* interface $I = \langle m, X \rangle$ *consists of a finite ordinal m called a* width*, a finite set X called a* name set*. An interface is* prime *if it has width* 1*.*

Definition A.4 (prime bigraph)**.** *A* prime *bigraph* $P : m \to \langle X \rangle$ *has no inner names and a prime outer face.*

Definition A.5 (place graph)**.** *A* place graph $A = (V, ctrl, prnt) : m \to n$ *has an* inner width *m and an* outer width *n, both finite ordinals; a finite set V of nodes with a control map* $ctrl : V \to \mathcal{K}$*; and a* parent map $prnt : m \uplus V \to V \uplus n$*. The parent map is* acyclic*, i.e.* $prnt^k(v) \neq v$ *for all* $k > 0$

and $v \in V$. An atomic *node – i.e. one whose control is atomic – may not be a parent. We write $w >_A w'$, or just $w > w'$, to mean $w = prnt^k(w')$ for some $k > 0$. The widths m and n index the* sites *and* roots *of A respectively. The sites and nodes – i.e. the domain of $prnt$ – are called* places.

Definition A.6 (precategory of place graphs)**.** *The precategory of place graphs* Plg *has finite ordinals as objects and place graphs as arrows. The composition $A_1 \circ A_0 : m_0 \to m_2$ of two place graphs $A_i = (V_i, ctrl_i, prnt_i) : m_i \to m_{i+1}$ $(i = 0, 1)$ is defined when the two node sets are disjoint; then $A_1 \circ A_0 \stackrel{\text{def}}{=} (V, ctrl, prnt)$ where $V = V_0 \uplus V_1$, $ctrl = ctrl_0 \uplus ctrl_1$, and $prnt = (\mathsf{Id}_{V_0} \uplus prnt_1) \circ (prnt_0 \uplus \mathsf{Id}_{V_1})$. The identity place graph at m is $\mathsf{id}_m \stackrel{\text{def}}{=} (\emptyset, \emptyset_{\mathcal{K}}, \mathsf{Id}_m) : m \to m$.*

Definition A.7 (barren,sibling,active,passive)**.** *A node or root is* barren *it is has no children. Two places are* siblings *if they have the same parent. A site s of A is* active *if $ctrl(v)$ is active whenever $v > s$; otherwise s is* passive*. If s is active (resp. passive) in A, we also say that A is* active *(resp.* passive*)* at *s.*

Definition A.8 (tensor product, Plg)**.** *The* tensor product $\otimes$ *in* Plg *is defined as follows: On objects, we take $m \otimes n = m + n$. For two place graphs $A_i : m_i \to n_i$ $(i = 0, 1)$ we take $A_0 \otimes A_1 : m_0 + m_1 \to n_0 + n_1$ to be defined when A_0 and A_1 have disjoint node sets; for the parent map, we first adjust the sites and roots of A_1 by adding them to m_0 and n_0 respectively, then take the union of the two parent maps.*

Definition A.9 (hard place graphs)**.** *A* hard *place graph is one in which no root or non-atomic node is barren. They form a sub-precategory denoted by* Plg_{h}.

Definition A.10 (link graph)**.** *A* link graph *$A = (V, E, ctrl, link) : X \to Y$ has finite sets X of* inner names*, Y of* (outer) names*, V of* nodes *and E of* edges*. It also has a function $ctrl : V \to \mathcal{K}$ called the* control map*, and a function $link : X \uplus P \to E \uplus Y$ called the* link map*, where $P \stackrel{\text{def}}{=} \sum_{v \in V} ar(ctrl(v))$ is the set of* ports *of A.*
We shall call the inner names X and ports P the points *of A, and the edges E and outer names Y its* links*.*

Definition A.11 (idle,open,closed,peer,lean)**.** *A link is* idle *if it has no preimage under the* link *map. An (outer) name is an* open *link, an edge is a* closed *link. A point (i.e. an inner name or port) is* open *if its link is open, otherwise* closed*. Two distinct points are* peers *if they are in the same link. A link graph is* lean *if it has no idle edges.*

Definition A.12 (precategory of link graphs)**.** *The precategory* Lig *has name sets as objects and link graphs as arrows. The composition* $A_1 \circ A_0 : X_0 \to X_2$ *of two link graphs* $A_i = (V_i, E_i, ctrl_i, link_i) : X_i \to X_{i+1}$ $(i = 0, 1)$ *is defined when their node sets and edge sets are disjoint; then* $A_1 \circ A_0 \stackrel{\text{def}}{=} (V, E, ctrl, link)$ *where* $V = V_0 \uplus V_1, ctrl = ctrl_0 \uplus ctrl_1, E = E_0 \uplus E_1$ *and* $link = (\mathsf{id}_{E_0} \uplus link_1) \circ (link_0 \uplus \mathsf{id}_{P_1})$*. The identity link graph at* X *is* $\mathsf{id}_X = (\emptyset, \emptyset, \emptyset_{\mathcal{K}}, \mathsf{id}_X) : X \to X$*.*

Definition A.13 (tensor product, Lig)**.** *The* tensor product $\otimes$ *in* Lig *is defined as follows: On objects,* $X \otimes Y$ *is simply the union of sets required to be disjoint. For two link graphs* $A_i : X_i \to Y_i$ $(i = 0, 1)$ *we take* $A_0 \otimes A_1 : X_0 \otimes X_1 \to Y_0 \otimes Y_1$ *to be defined when the interface products are defined and when* A_0 *and* A_1 *have disjoint node sets and edge sets; then we take the union of their link maps.*

Definition A.14 (parallel product)**.** *The* parallel product $\|$ *in* Lig *is defined as follows: On objects,* $X \| Y \stackrel{\text{def}}{=} X \cup Y$*. On link graphs* $A_i : X_i \to Y_i$ $(i = 0, 1)$ *we define* $A_0 \| A_1 : X_0 \otimes X_1 \to Y_0 \| Y_1$ *whenever* X_0 *and* X_1 *are disjoint, by taking the union of link maps.*

A place graph can be combined with a link graph iff they have the same node set and control map.

Definition A.15 (precategory of pure concrete bigraphs)**.** *The precategory* $\mathcal{B}\text{ig}(\mathcal{K})$ *of pure concrete bigraphs over a signature* $\mathcal{K}$ *has pairs* $I = \langle m, X \rangle$ *as objects (*interfaces*) and bigraphs* $G = (V, E, ctrl_G, G^{\mathsf{P}}, G^{\mathsf{L}}) : I \to J$ *as arrows (*contexts*). We call* I *the* inner face *of* G*, and* I *the* outer face*. If* $H : J \to K$ *is another bigraph with node set disjoint from* V*, then their composition is defined directly in terms of the compositions of the constituents as follows:*

$$H \circ G \stackrel{\text{def}}{=} \langle H^{\mathsf{P}} \circ G^{\mathsf{P}}, H^{\mathsf{L}} \circ G^{\mathsf{L}} \rangle : I \to K.$$

The identities are $\langle \mathsf{id}_m, \mathsf{id}_X \rangle : I \to I$*, where* $I = \langle m, X \rangle$*.*
The subprecategory $\mathcal{B}\text{ig}_{\mathsf{h}}$ *consists of* hard *bigraphs, those with place graphs in* $\mathcal{P}\text{lg}_{\mathsf{h}}$*.*

Definition A.16 (tensor product, Big)**.** *The* tensor product *of two bigraph interfaces is defined by* $\langle m, X \rangle \otimes \langle n, Y \rangle \stackrel{\text{def}}{=} \langle m + n, X \cup Y \rangle$ *when* X *and* Y *are disjoint. The* tensor product *of two bigraphs* $G_i : I_i \to J_i$ $(i = 0, 1)$ *is defined by*

$$G_0 \otimes G_1 \stackrel{\text{def}}{=} \langle G_0^{\mathsf{P}} \otimes G_1^{\mathsf{P}}, G_0^{\mathsf{L}} \otimes G_1^{\mathsf{L}} \rangle : I_0 \otimes I_1 \to J_0 \mid J_1$$

when the interfaces exist and the node sets are disjoint. This combination is well-formed, since its constituents share the same node set.

Definition A.17 (parallel product, ´Big)**.** *The* parallel product *of two bigraphs is defined on interfaces by* $\langle m, X\rangle \,|\langle\, n, Y\rangle \stackrel{\text{def}}{=} \langle m+n, X \cup Y\rangle$*, and on bigraphs by*

$$G_0 \mid G_1 \stackrel{\text{def}}{=} \langle G_0^P \otimes G_1^P, G_0^L \parallel G_1^L\rangle : I_o \otimes I_1 \to J_0 \mid J_1$$

when the interfaces exist and the node sets are disjoint.

It is easy to verify that $\parallel$ is associative, with unit ϵ.

Proposition A.18 (alternative parallel product, ´Big)**.** *Let* $G_0 \parallel G_1$ *be defined. Then*

$$G_0 \parallel G_1 = \sigma(G_0 \otimes \tau G_1)\,,$$

where the substitutions σ *and* τ *are defined as follows: If* $z_i (i \in n)$ *are the names shared between* G_0 *and* G_1*, and* w_i *are fresh names in bijection with the* z_i*, then* $\tau(z_i) = w_i$ *and* $\sigma(w_i) = \sigma(z_i) = z_i (i \in n)$*.*

Definition A.19 (prime product, ´Big)**.** *The* prime product *of two interfaces is given by*

$$\langle m, X\rangle \,|\langle\, n, Y\rangle \stackrel{\text{def}}{=} \langle 1, X \cup Y\rangle.$$

For two prime bigraphs $\vec{P} : \vec{I} \to \vec{J}$*, if* $I_0 \otimes I_1$ *defined and* n *is the sum of the widths of* J_0 *and* J_1*, we define their* prime product *by*

$$P_0 \mid P_1 \stackrel{\text{def}}{=} merge_n \circ (P_0 \parallel P_1) : I_0 \otimes I_1 \to J_0 \mid J_1\,.$$

Again $\mid$ is associative, with unit 1 when applied to primes.

Definition A.20 (s-category)**.** *An* s-category *C is a strict symmetric monoidal precategory which has:*
– for each arrow f*, a finite set* $|f|$ *called its* support*, such that* $|\mathsf{id}_I| = \emptyset$*. For* $f : I \to J$ *and* $g : J \to K$ *the composition* $gf : I \to K$ *is defined iff* $|g| \cap |f| = \emptyset$ *and* $dom(g) = cod(f)$*; then* $|gf| = |g| \uplus |f|$*. Similarly, for* $f : H \to I$ *and* $g : J \to K$ *with* $H \otimes J$ *and* $I \otimes K$ *defined, the tensor product* $f \otimes g : H \otimes J \to I \otimes K$ *is defined iff* $|f| \cap |g| = \emptyset$*; then* $|f \otimes g| = |f| \uplus |g|$*.*
– for any arrow $f : I \to J$ *and any injective map* ρ *whose domain includes* $|f|$*, an arrow* $f : I \to J$ *called a* support translation *of* f *such that*

1. $\rho \cdot \mathsf{id}_I = \mathsf{id}_I$

2. $\rho \cdot (gf) = (\rho \cdot g)(\rho \cdot f)$
3. $\rho \cdot (f \otimes g) = \rho \cdot f \otimes \rho \cdot g$
4. $\mathsf{Id}_{|f|} \cdot f = f$
5. $(\rho_1 \circ \rho_0) \cdot f = \rho_1 \cdot (\rho_0 \cdot f)$
6. $\rho \cdot f = (\rho | |f|) \cdot f$
7. $|\rho \cdot f| = \rho(|f|)$.

Each equation is required to hold only when both sides are defined.

Definition A.21 (support equivalence,supported functor)**.** *Let* ´A *be a supported precategory. Two arrows* $f, g : I \to J$ *in* ´A *are* support-equivalent*, written* $f \simeq g$*, if* $\rho \cdot f = g$ *for some support translation* ρ*. By Definition A.20(6) and (7)this is an equivalence relation. If* ´B *is another supported precategory, then a functor* $\mathcal{F} : \text{´A} \to \text{´B}$ *is called* supported *if it preserves support equivalence, i.e.* $f \simeq g \implies \mathcal{F}(f) \simeq \mathcal{F}(g)$*.*

Definition A.22 (instantiation)**.** *An* instantiation ϱ *from (width)* m *to (width)* n*, which we write* $\varrho :: m \to n$*, is determined by function* $\overline{\varrho} : n \to m$*. For any* X *this function defines the map*

$$\varrho : \mathrm{Gr}\langle m, X\rangle \to \mathrm{Gr}\langle n, X\rangle$$

as follows. Decompose $g : \langle m, X\rangle$ *into* $g = w(d_0 \otimes \cdots \otimes d_{m-1})$*, with* $w : Y \to X$ *and each* d_i *prime and discrete. Then define*

$$\varrho(g) \stackrel{\text{def}}{=} w(e_0 \parallel \cdots \parallel e_{n-1}),$$

where $e_j \simeq d_{\overline{\varrho}(j)}$ *for* $j \in n$*. This map is well-defined (up to support translation), by Propositions 9.16 and 9.17.*

Note that the names of $e_0 \parallel \cdots \parallel e_{n-1}$ may be fewer than Y, because ϱ may not be surjective. But by our convention the outer names of $\varrho(g)$ are determined by the outer names of w, i.e. X.

Definition A.23 (binding signature)**.** *A* binding signature $\mathcal{K}$ *is like a pure signature, except that the arity of a control* $K : h \to k$ *now consists of a pair of finite ordinals: the* binding arity h *and the* free arity k*, determining the number of* binding *and* non-binding *ports of any K-node. If K is atomic then* $h = 0$*.*

Definition A.24 (binding interface)**.** *A* binding interface $I = \langle m, loc, X\rangle$*, where the width* m *is as before,* X *is a finite set of names, and* $loc : X \to m \uplus \{\bot\}$ *is a* locality *map associating some of the names* X *with a site in* m*. If* $loc(x) = s \in m$ *then* x *is* located *at* s*, or* local *(to* s*); If* $loc(x) = \bot$ *then* x *is* global*.*
We call $I^U = \langle m, X\rangle$ *the pure interface* underlying I*.*

Definition A.25 (binding bigraphs)**.** *A* (concrete) binding bigraph $G : I \to J$ *consists of an* underlying *pure bigraph* $G^u : I^u \to J^u$ *with extra structure as follows. Declare its* binders *to be the binding ports of its nodes together with the local names of its outer face* J*. Then* G *must satisfy the following:*

> Scope rule: *If* p *is a binder located at a node or root* w*, then every peer* p' *of* p *must be located at a place* w' *(a site or node) such that* $w' <_{G^u} w$*.*

In the precategory $\acute{}\textsc{Bbg}(\mathcal{K})$ *of (concrete) binding bigraphs over* $\mathcal{K}$*, composition and identities are defined as for the underlying pure bigraphs; they are easily found to respect the scope rule. The forgetful functor*

$$\mathcal{U} : \acute{}\textsc{Bbg}(\mathcal{K}) \to \acute{}\textsc{Big}(\mathcal{K})$$

sends each I *to* I^u *and each* G *to* G^u*. The analogous definition holds also for hard binding bigraphs* $\acute{}\textsc{Bbg}_{\mathsf{h}}(\mathcal{K})$*.*

Definition A.26 (tensor product, $\acute{}\textsc{Bbg}$)**.** *The tensor product of interfaces* $I = \langle m, \vec{X}, X\rangle$ *and* $J = \langle n, \vec{Y}, Y\rangle$*, where* X *and* Y *are disjoint, is*

$$I \otimes J = \langle m + n, \vec{X}\vec{Y}, X \uplus Y\rangle.$$

The tensor product $G : I \to J$ *of two binding* $G_i : I_i \to J_i (i = 0, 1)$ *with disjoint supports is defined when* $I = I_0 \otimes I_1$ *and* $J = J_0 \otimes J_1$ *are defined, and then* $G^u = G_0^u \otimes G_1^u$*. Thus* $\mathcal{U}$ *preserves tensor product.*

Definition A.27 (parallel product, $\acute{}\textsc{Bbg}$)**.** *Extending the previous definition, the parallel product of two interfaces* $J_i = \langle n_i, \vec{X}_i, Y_i\rangle (i = 0, 1)$ *keeps their local names disjoint but may share their global names:*

$$J_0 \parallel J_1 \stackrel{\text{def}}{=} \langle n_0 + n_1, \vec{X}_0\vec{X}_1, Y_0 \cup Y_1\rangle\ .$$

We define a parallel product on binding bigraphs by the equation $G_0 \parallel G_1 = \sigma(G_0 \otimes \tau G_1)$*.*

Definition A.28 (prime product, ´Bbg)**.** *Extending the previous definition, the prime product of two prime interfaces is*

$$\langle (X'), X\rangle \mid\langle\ (Y'), Y\rangle \stackrel{\text{def}}{=} \langle (X' \uplus Y'), X \cup Y\rangle\ .$$

The expression of the prime product of two prime binding bigraphs in terms of their parallel product is just as before.

Definition A.29 (instantiation, ´Bbg)**.** *We replace instantiations $\varrho :: m \to n$ for pure bigraphs by instantiations $\varrho :: I \to J$ for binding bigraphs, where $I = \langle m, \vec{X}\rangle$ and $J = \langle n, \vec{Y}\rangle$ are local. The instantiation consists again of an underlying function $\overline{\varrho} : n \to m$, and also provides bijective local substitutions $\varrho_j : (X_{\overline{\varrho}(j)}) \to (Y_j)$ for all $j \in n$. These ensure disjoint local names for each copy of a parameter factor. For any Z, this allows the map*

$$\varrho : \mathrm{Gr}(I \otimes Z) \to \mathrm{Gr}(J \otimes Z)$$

to be defined as follows (in terms of DNF as before): Decompose $g : I \otimes Z$ into $g = w(d_0 \otimes \cdots \otimes d_{m_1})$ with $w : W \to Z$ and each d_i prime and discrete. Then let $e_j \eqsim \varrho_j \circ d_{\overline{\varrho}(j)}$ for each $j \in n$, and define

$$\varrho(g) \stackrel{\text{def}}{=} w(e_0 \parallel \cdots \parallel e_{n-1})\ .$$

Definition A.30 (bigraphical reactive system)**.** *A* bigraphical reactive system (BRS) *over signature $\mathcal{K}$ consists of ´Bbg($\mathcal{K}$) equipped with a set ´*Reacts *of reaction rules closed under support equivalence ($\eqsim$). We denote it by ´Bbg($\mathcal{K}$, ´*Reacts*).*

Definition A.31 (Insertion, [Jen07])**.** *Given a wiring $\omega : X \to Y$ and a local prime $A : X' \to Y'$ the* insertion *of ω into A is defined iff X and X' are disjoint. The result, written $A \lhd \omega : XX' \to Y \cup Y'$, has the nodes and parent map of A and its link map is the union of those of A and ω. Insertion binds tighter than prime product and composition.*

We continue on the next page with the axioms for Binding Bigraphs.

The axioms for binding bigraphs in Def. A.32 are from [DB06], but with explicit composition. A, B, C, G range over bigraphs, H, I, J, K range over interfaces, ϵ is the empty interface $\langle 0, (), \emptyset \rangle$, x, y range over names, X, Y, Z range over name sets,
$\ulcorner X \urcorner^Z \stackrel{\text{def}}{=} (Z)\ulcorner Z \uplus X \urcorner : \langle 1, (Z \uplus X), Z \uplus X \rangle \to \langle 1, (Z), Z \uplus X \rangle$, P ranges over primes, $\mathsf{K}_{\vec{(}y)(\vec{X})}$ over ions, α ranges over renamings (multiple bijective substitutions), σ ranges over substitutions, and σ^{loc} ranges over local substitutions.

Definition A.32 (Axioms for binding bigraphs)**.**

Categorical axioms

(C1)	$A \circ \mathsf{id}_I$	$=$	A	$= \mathsf{id}_J \circ A \qquad (A : I \to J)$
(C2)	$A \circ (B \circ C)$	$=$	$(A \circ B) \circ C$	
(C3)	$A \otimes \mathsf{id}_\epsilon$	$=$	A	$= \mathsf{id}_\epsilon \otimes A$
(C4)	$A \otimes (B \otimes C)$	$=$	$(A \otimes B) \otimes C$	
(C5)	$\mathsf{id}_I \otimes \mathsf{id}_J$	$=$	$\mathsf{id}_{I \otimes J}$	
(C6)	$(A_1 \otimes B_1) \circ (A_0 \otimes B_0)$	$=$	$(A_1 \circ A_0) \otimes (B_1 \circ B_0)$	
(C7)	$\gamma_{I,\epsilon}$	$=$	id_I	
(C8)	$\gamma_{J,I} \circ \gamma_{I,J}$	$=$	$\mathsf{id}_{I \otimes J}$	
(C9)	$\gamma_{I,K} \circ (A \otimes B)$	$=$	$(B \otimes A) \circ \gamma_{H,J} \quad (A : H \to I, B : J \to K)$	
(C10)	$\gamma_{I \otimes J,K} \circ (A \otimes B)$	$=$	$(\gamma_{I,K} \otimes \mathsf{id}_J) \circ (\mathsf{id}_I \otimes \gamma_{J,K})$	

Link axioms

(L1)	$x/_x$	$=$	id_x
(L2)	$/_y \circ y/_x$	$=$	$/_x$
(L3)	$/_y \circ y$	$=$	id_ϵ
(L4)	$(z/_{(Y \uplus y)}) \circ (\mathsf{id}_Y \otimes y/_X)$	$=$	$z/_{(Y \uplus X)}$

Place axioms

(P1)	$join \circ (1 \otimes \mathsf{id}_1)$	$=$	id_1
(P2)	$join \circ (join \otimes \mathsf{id}_1)$	$=$	$join \circ (\mathsf{id}_1 \otimes join)$
(P3)	$join \circ \gamma_{1,1,(\emptyset,\emptyset)}$	$=$	$join$

Binding axioms

(B1)	$(\emptyset)P$	$=$	P
(B2)	$(Y)\ulcorner Y \urcorner$	$=$	$\mathsf{id}_{(Y)}$
(B3)	$(\ulcorner X \urcorner^Z \otimes \mathsf{id}_Y)(X)P$	$=$	$P \qquad (P : I \to \langle 1, (Z), Z \uplus X \uplus Y \rangle)$
(B4)	$(((Y)P) \otimes \mathsf{id}_X)G$	$=$	$(Y)(P \otimes \mathsf{id}_X)G$
(B5)	$(X \uplus Y)P$	$=$	$(X)((Y)P)$

Ion axioms

(N1)	$(\mathsf{id}_1 \otimes \alpha) \circ \mathsf{K}_{\vec{y}(\vec{X})}$	$=$	$\mathsf{K}_{\alpha(\vec{y})(\vec{X})}$
(N2)	$\mathsf{K}_{\vec{y}(\vec{X})} \circ \sigma^{\text{loc}}$	$=$	$\mathsf{K}_{\vec{y}((\sigma^{\text{loc}})^{-1}(\vec{X}))}$

A.2 Code

A.2.1 Simulation of an abstract location model

```
(*************************************************************
 Ebbe Elsborg
 -------------------------------------------------------------

 An abstract Plato-graphical location model.
 Simulation of a simple predefined scenario.

 -------------------------------------------------------------

 C: Real world

 signature world =
   sig %
     loc : passive (0)
     dev : passive (0)
     id  : passive (0)
     i0,i1,... : atomic
   end

 using world

 rule moveup  = (* sort: C *)
   loc(id([0]) | [1] | loc(id([2]) | [3] | dev([4])))
     ->
   loc(id([0]) | [1] | loc(id([2]) | [3]) | dev([4]))

 rule movedown = (* sort: C *)
   loc(id([0]) | [1] | loc(id([2]) | [3]) | dev([4]))
     ->
   loc(id([0]) | [1] | loc(id([2]) | [3] | dev([4])))

 state = loc(id(n) | loc(...) | dev(id(m)) | ...)

 -------------------------------------------------------------

 S: Sensor system
```

```
 signature sensor =
 sig end

(* rule schema, one for each n in i0,01,i2... *)
rule observe_update = (* sort: C,L *)
  loc(id([0]) | [1] | dev(n)) ||
  devs(location(l([2]) | d(n)) | [3])
    ->
  loc(id([0]) | [1] | dev(n)) ||
  devs(location(l([0]) | d(n)) | [3])

(* need a NAC to ensure that
   the iN node in [2] is not in [3] *)
rule observe_new = (* sort: C,L *)
  loc(id([0]) | [1] | dev([2])) ||
  devs([3])
    ->
  loc(id([0]) | [1] | dev([2])) ||
  devs([3] | location(l([0]) | d([2])))

(* need a NAC to ensure that
   the iN node in [3] is not in [0] *)
rule lose = (* sort: C,L *)
  loc([0]) || devs([1] | location(l([2]) | d([3])))
    ->
  loc([0]) || devs([1])

-------------------------------------------------------------

L: Location model

signature repr =
  sig
    devs     : passive (0)
    location : passive (0)
    l: passive (0)
    d: passive(0)
  end

state = devs(location(l(n),d(m)) | ...)
```

```
-------------------------------------------------------------

A: Location-aware application

signature agent =
 sig
   findall  : atomic
   whereis  : passive (0)
   i0,i1,i2... : atomic
   id : passive (0)
   location : passive (0)
   l: passive(0)
   d: passive(0)
 end

using agent

rule findall = (* sort: L,A *)
  devs([0]) || findall
    ->
  devs([0]) || [0]

rule whereis = (* sort: L,A *)
  devs(location(l([0]) | d(id([1])) | [2])) || whereis([1])
    ->
  devs(location(l([0]) | d(id([1])) | [2]))
  || location(l([0]) | d(id([1])))

rule genFindall = (* sort: A *)
  [0] -> [0] | findall

(* rule schema, a rule for each n in i0,01,i2... *)
rule genWhereis = (* sort: A *)
  [0] -> [0] | whereis(n)

-------------------------------------------------------------

INITIAL STATE OF THE SYSTEM:
  C: loc(id(i1) | loc(id(i2) | dev(i3) | dev(i4))) ||
```

```
   L: devs(location(l(i2) | d(i3))) ||
   A: 1

 ------------------------------------------------------------

 SCENARIO:
 1. A ``whereis'' query for device i3 is issued.
 2. A move of device i3 occurs in C.
 3. An answer to the query appears in A.
 4. Another ``whereis'' query is issued, now for device i4.
 5. S discovers that dev. i4 is in C but not in L and reacts.
 6. An answer to this query appears in A.

 EXTENDED:
 7. A ``findall'' query is issued.
 8. An answer to this query appears in A.

**************************************************************)

structure BG = BG (structure ErrorHandler = PrintErrorHandler)
structure B = BG.BgVal
structure S = BG.Sugar
structure P = BG.Permutation
structure R = BG.Rule
structure Bdnf = BG.BgBDNF
structure M = BG.Match
structure C = BG.Control
structure Name = BG.Name
structure NameSet = BG.NameSet
structure Ion = BG.Ion
structure Wiring = BG.Wiring
structure Link = BG.Link
structure Inst = BG.Instantiation
structure Iface = BG.Interface
structure Re = Reaction
  (structure RuleNameMap = Util.StringMap
   structure Info = BG.Info
   structure Interface = BG.Interface
   structure Wiring = BG.Wiring
   structure BgVal = BG.BgVal
```

```
   structure BgBDNF = BG.BgBDNF
   structure Match = BG.Match
   structure Instantiation = BG.Instantiation
   structure Rule = BG.Rule
   structure Origin = Origin
   structure ErrorHandler = PrintErrorHandler)

(* make look nicer *)
val _ = Flags.setBoolFlag "/kernel/ast/bgterm/ppids" false
val _ = Flags.setBoolFlag "/kernel/ast/bgterm/ppabs" false
val _ = Flags.setBoolFlag "/kernel/ast/bgterm/pp0abs" false
val _ = Flags.setBoolFlag "/kernel/ast/bgterm/pptenaspar" true
val _ = Flags.setBoolFlag "/kernel/ast/bgterm/ppmeraspri" true
val _ = Flags.setBoolFlag "/kernel/ast/bgval/pp-simplify" true

(* useful constants *)
val info = BG.Info.noinfo
val barren = S.<->
val id_1 = B.Per info (P.id_n 1)
val site = id_1 (*‘[]‘*)

(* exception handler *)
fun handler exn =
  (print (BaseErrorHandler.explain' exn) ; print "\n")

(* shorthand functions *)
fun s2n s = Name.make s
fun n2s n = Name.unmk n
fun v2n x = Name.make (String.toString x)
fun ion2bg ion = B.Ion info ion

(* interface functions *)
fun getInner b = let val (b,inn,out) = B.unmk b in inn end
fun getOuter b = let val (b,inn,out) = B.unmk b in out end
fun isGround b = let val (bgterm,inner,outer) = B.unmk b
 in Iface.eq (inner, Iface.zero) end

(* LazyList functions*)
fun lzLength l =
  Int.toString(List.length(LazyList.lztolist l))
```

```
(* take apart a match; context and parameter *)
fun parts agent matches =
    let val agent' = M.unmk (LazyList.lzhd matches)
val agent'_ctx = #context(agent')
val agent'_par = #parameter(agent')
fun peel x = (B.toString o B.simplify o Bdnf.unmk) x
    in ["agent_ctx= " ^ (peel agent'_ctx) ^ "\n",
"agent_par= " ^ (peel agent'_par) ^ "\n"] end

(* printing *)
fun printWidth w = print(Int.toString w)
fun printName n = print(", " ^ (n2s n))
fun printNameset set =
    let fun loop set strAcc flag =
    if NameSet.size set = 0 then strAcc
    else let val member = NameSet.someElement set
     val set' = NameSet.remove member set
 in if flag (* at first member *)
    then
                      loop set' (strAcc ^ (n2s member)) false
    else
                      loop set'
                           (strAcc ^ "," ^ (n2s member))
                           false
 end
    in print ("{" ^ (loop set "" true) ^ "}") end

fun printLoc l =
    let fun loop list flag =
    case list
     of [] => print ""
      | (x::xs) =>
                  if flag
                  then ( printNameset x ; loop xs false )
          else ( print ","
                       ; printNameset x
                       ; loop xs false )
    in ( print "(" ; loop l true ; print ")") end
```

```
fun printGlob s = printNameset s

fun printIface i =
    let val i_parts = Iface.unmk i
val w = #width(i_parts)
val l = #loc(i_parts)
val g = #glob(i_parts)
    in ( print "<"
       ; printWidth w
       ; print ", "
       ; printLoc l
       ; print ", "
       ; printGlob g
       ; print ">") end

fun printIfaces t i j =
    ( print(t ^ " : ")
    ; printIface i
    ; print " -> "
    ; printIface j
    ; print "\n")

fun prtSimp name bgval =
    print(name ^ "= " ^ B.toString(B.simplify bgval) ^ "\n")

fun printMts m =
    ( print "Matches:\n"
    ; LazyList.lzprint M.toString m
    ; print "\n" )

fun printRes tname agents =
    ( print("\nAgent(s) resulting from reaction(s) on "
      ^ tname ^ ":\n")
    ; LazyList.lzprint (B.toString o B.simplify) agents
    ; print "\n" )

fun printRes' tname rname agent =
    ( print("Agent resulting from reaction with "
    ^ rname ^ " on " ^ tname ^ ":\n")
    ; print((B.toString o B.simplify) agent)
```

```
    ; print "\n" )


(* SIGNATURE *)

(* C *)
fun i n = S.atomic0 ("i" ^ n)
val id = S.passive0 "id"
val loc = S.active0 "loc"
val dev = S.passive0 "dev"

(* S has the empty signature *)

(* L *)
val l = S.passive0 "l"
val d = S.passive0 "d"
val devs = S.passive0 "devs"
val location = S.passive0 "location"

(* A *)
val findall = S.atomic0 "findall"
val whereis = S.passive0 "whereis"

(* A shares 'fun i' and 'val id' with C. *)
(* A shares 'val l', 'val d', and 'val location' with L. *)

(* Plato-graphical systems *)
fun make_plato (c,p,a) = S.|| (c, S.|| (p, a))

(* auxiliary definitions *)
val loc' = S.o (loc, S.`|` (id, site))
fun loc'' n = S.o (loc, S.`|` (S.o (id, i(n)), site))
fun dev' n = S.o (dev, i(n))
fun devs' h = S.o (S.passive0 "devs", h)
val locdev = S.o (loc, S.`|` (id, S.`|` (site, dev)))
fun locdev' n = S.o (loc, S.`|` (id, S.`|` (site, dev' n)))
val location' = S.o (location, S.`|` (l, d))
fun location'' lid did =
    S.o (location, S.`|` (S.o (l, i(lid)), S.o (d, i(did))))
fun location''' did =
```

```
    S.o (location, S.‘|‘ (l, S.o (d, i(did))))
fun whereis’ c = S.o (S.passive0 "whereis", c)

(* initial state of the system *)
val loc1 = loc’’ "1"
val loc2 = loc’’ "2"
val dev3 = dev’ "3"
val dev4 = dev’ "4"
val C = S.o (loc1, S.o (loc2, S.‘|‘ (dev3, dev4)))
val location_l2d3 = location’’ "2" "3"
val L = S.o (devs, location_l2d3)
val A = barren
val system0 = make_plato(C,L,A)

val _ = prtSimp "\nsystem0\n" system0
val _ = printIfaces
          "system0" (getInner system0) (getOuter system0)

(* aux. function *)
fun makeBR bgval = Bdnf.regularize (Bdnf.make bgval)


(* RULES *)

(* C *)

val aux1 =
  S.o (loc, S.‘|‘(id, S.‘|‘(site,
    S.o (loc, S.‘|‘(id, S.‘|‘(site, S.‘|‘(dev, dev)))))))
val aux2 =
  S.o (loc, S.‘|‘(id, S.‘|‘(site,
    S.‘|‘(S.o (loc, S.‘|‘(id, S.‘|‘ (site, dev))), dev))))
val redex_innerface_move = Iface.m 6
val react_innerface_move = Iface.m 6
val instMove = Inst.make { I = redex_innerface_move,
   J = react_innerface_move,
   maps = [((0,[]), (0,[])),
   ((1,[]), (1,[])),
   ((2,[]), (2,[])),
   ((3,[]), (3,[])),
```

```
   ((4,[]), (5,[])),
   ((5,[]), (4,[]))] }

val redexUp = aux1
val reactUp = aux2
val Cmoveup = R.make { name = "Cmoveup",
       redex = makeBR redexUp,
       react = reactUp,
       inst = instMove,
       info = info }

val redexDown = aux2
val reactDown = aux1
val Cmovedown = R.make { name = "Cmovedown",
 redex = makeBR redexDown,
 react = reactDown,
 inst = instMove,
 info = info }

(* S *)
(* rule schema, one for each n in N *)
val devs_dn = S.o (devs, S.`|` (location''' "n", site))
val redexObsUpd = S.|| (locdev' "n", devs_dn)
val reactObsUpd = redexObsUpd
val redex_innerface_upd = Iface.m 4
val react_innerface_upd = Iface.m 4
val instUpd = Inst.make { I = redex_innerface_upd,
  J = react_innerface_upd,
  maps = [((0,[]), (0,[])),
  ((1,[]), (1,[])),
  ((2,[]), (0,[])),
  ((3,[]), (3,[]))] }
val Sobsupd = R.make { name = "Sobsupd",
       redex = makeBR redexObsUpd,
       react = reactObsUpd,
       inst = instUpd,
       info = info }

(* need a NAC to ensure that iN of dev is not in devs *)
val redexObsNew = S.|| (locdev, devs)
```

```
val reactObsNew =
  S.||(locdev, devs' (S.`|`(site, location')))
val redex_innerface_new = Iface.m 4
val react_innerface_new = Iface.m 6
val instNew = Inst.make { I = redex_innerface_new,
  J = react_innerface_new,
  maps = [((0,[]), (0,[])),
  ((1,[]), (1,[])),
  ((2,[]), (2,[])),
  ((3,[]), (3,[])),
  ((4,[]), (0,[])),
  ((5,[]), (2,[]))] }
val Sobsnew = R.make { name = "Sobsnew",
       redex = makeBR redexObsNew,
       react = reactObsNew,
       inst = instNew,
       info = info }

(* need a NAC to ensure that
   site in dev is not in site of loc *)
val redexLose = S.|| (loc, devs' (S.`|` (site, location')))
val reactLose = S.|| (loc, devs)
val Slose = R.make' { name = "Slose",
      redex = makeBR redexLose,
      react = reactLose,
      info = info }

(* L has no rules *)

(* A *)
val redex_innerface_findall = Iface.m 1
val react_innerface_findall = Iface.m 2
val instFindall = Inst.make { I = redex_innerface_findall,
      J = react_innerface_findall,
      maps = [((0,[]), (0,[])),
      ((1,[]), (0,[]))] }
val redexFindall = S.|| (devs, findall)
val reactFindall = S.|| (devs, site)
val Afindall = R.make { name = "Afindall",
redex = makeBR redexFindall,
```

```
react = reactFindall,
inst = instFindall,
info = info }
(*    handle e => (handler e ;
   R.make' { name = "dummy",
     redex = makeBR barren,
     react = barren,
     info = info })*)

(* rule schema, a rule for each n in N,
   we need two concrete now *)
val redexWhereis3 =
  S.|| (devs' (S.`|` (location''' "3", site)),
        whereis'(i("3")))
val reactWhereis3 =
  S.|| (devs' (S.`|` (location''' "3", site)),
location''' "3")
val redexWhereis4 =
  S.|| (devs' (S.`|` (location''' "4", site)),
whereis'(i("4")))
val reactWhereis4 =
  S.|| (devs' (S.`|` (location''' "4", site)),
location''' "4")
val redex_innerface_whereis = Iface.m 2
val react_innerface_whereis = Iface.m 3
val instWhereis = Inst.make { I = redex_innerface_whereis,
      J = react_innerface_whereis,
      maps = [((0,[]), (0,[])),
      ((1,[]), (1,[])),
      ((2,[]), (0,[]))] }
val Awhereis3 = R.make { name = "Awhereis3",
 redex = makeBR redexWhereis3,
 react = reactWhereis3,
 inst = instWhereis,
 info = info }
val Awhereis4 = R.make { name = "Awhereis4",
 redex = makeBR redexWhereis4,
 react = reactWhereis4,
 inst = instWhereis,
 info = info }
```

```
val redexGenFindall = site
val reactGenFindall = S.‘|‘ (site, findall)
val AgenFindall = R.make’ { name = "AgenFindall",
    redex = makeBR redexGenFindall,
    react = reactGenFindall,
    info = info }

(* rule schema, a rule for each n in N,
   we need two concrete now *)
val redexGenWhereis3 = site
val reactGenWhereis3 = S.‘|‘ (site, whereis’(i("3")))
val AgenWhereis3 = R.make’ { name = "AgenWhereis3",
     redex = makeBR redexGenWhereis3,
     react = reactGenWhereis3,
     info = info }
val redexGenWhereis4 = site
val reactGenWhereis4 = S.‘|‘ (site, whereis’(i("4")))
val AgenWhereis4 = R.make’ { name = "AgenWhereis4",
     redex = makeBR redexGenWhereis4,
     react = reactGenWhereis4,
     info = info }

(* REACTIONS *)

(* Scenario:
 loc(id(i1) | loc(id(i2) | dev(i3) | dev(i4))) ||
 devs(location(l(i2) | d(i3))) ||
 1
   --1:Agenwhereis3->
 loc(id(i1) | loc(id(i2) | dev(i3) | dev(i4))) ||
 devs(location(l(i2) | d(i3))) ||
 whereis(i3)
   --2:Cmoveup->
 loc(id(i1) | dev(i3) | loc(id(i2) | dev(i4))) ||
 devs(location(l(i2) | d(i3))) ||
 whereis(i3)
   --3:Awhereis3->
 loc(id(i1) | dev(i3) | loc(id(i2) | dev(i4))) ||
 devs(location(l(i2) | d(i3))) ||
```

```
 location(l(i2) | d(i3))
   --4:AgenWhereis4->
 loc(id(i1) | dev(i3) | loc(id(i2) | dev(i4))) ||
 devs(location(l(i2) | d(i3))) ||
 (location(l(i2) | d(i3)) | whereis(i4))
   --5:Sobsnew->
 loc(id(i1) | dev(i3) | loc(id(i2) | dev(i4))) ||
 devs(location(l(i2) | d(i3)) | location(l(i1) | d(i4))) ||
 (location(l(i2) | d(i3)) | whereis(i4))
   --6:Awhereis4->
 loc(id(i1) | dev(i3) | loc(id(i2) | dev(i4))) ||
 devs(location(l(i2) | d(i3)) | location(l(i1) | d(i4))) ||
 (location(l(i2) | d(i3)) | location(l(i1) | d(i4)))
   --7:AgenFindall->
 loc(id(i1) | dev(i3) | loc(id(i2) | dev(i4))) ||
 devs(location(l(i2) | d(i3)) | location(l(i1) | d(i4))) ||
 (location(l(i2) | d(i3)) | location(l(i1) | d(i4)) | findall)
   --8:Afindall->
 loc(id(i1) | dev(i3) | loc(id(i2) | dev(i4))) ||
 devs(location(l(i2) | d(i3)) | location(l(i1) | d(i4))) ||
 (location(l(i2) | d(i3)) | location(l(i1) | d(i4)) |
  location(l(i2) | d(i4)) | location(l(i2) | d(i3)))
*)

(* 1: --AgenWhereis-> *)
val BRsystem0 = makeBR system0
val mts0 = M.matches {agent = BRsystem0, rule = AgenWhereis3}
val match0 = LazyList.lznth mts0 16 (* zero-indexed *)
val system1 = Re.react match0 (* return agent *)
val _ = print "\n"
val _ = printRes' "system0" "AgenWhereis" system1

(* 2: --Cmoveup-> *)
val BRsystem1 = makeBR system1
val mts1 = M.matches { agent = BRsystem1 , rule = Cmoveup }
val match1 = LazyList.lznth mts1 0
val system2 = Re.react match1
val _ = print "\n"
val _ = printRes' "system1" "Cmoveup" system2
```

```
(* 3: --Awhereis-> *)
val BRsystem2 = makeBR system2
val mts2 = M.matches { agent = BRsystem2 , rule = Awhereis3 }
val match2 = LazyList.lznth mts2 0
val system3 = Re.react match2
val _ = print "\n"
val _ = printRes' "system2" "Awhereis" system3

(* 4: --AgenWhereis-> *)
val BRsystem3 = makeBR system3
val mts3 = M.matches { agent = BRsystem3 ,
                       rule = AgenWhereis4 }
val match3 = LazyList.lznth mts3 17
val system4 = Re.react match3
val _ = print "\n"
val _ = printRes' "system3" "AgenWhereis" system4

(* 5: --Sobsnew-> *)
val BRsystem4 = makeBR system4
val mts4 = M.matches { agent = BRsystem4 , rule = Sobsnew }
val match4 = LazyList.lznth mts4 1
val system5 = Re.react match4
val _ = print "\n"
val _ = printRes' "system4" "Sobsnew" system5

(* 6: --Awhereis-> *)
val BRsystem5 = makeBR system5
val mts5 = M.matches { agent = BRsystem5 , rule = Awhereis4 }
val match5 = LazyList.lznth mts5 0
val system6 = Re.react match5
val _ = print "\n"
val _ = printRes' "system5" "Awhereis" system6

(* 7: --AgenFindall-> *)
val BRsystem6 = makeBR system6
val mts6 = M.matches { agent = BRsystem6 ,
                       rule = AgenFindall }
val match6 = LazyList.lznth mts6 19
val system7 = Re.react match6
val _ = print "\n"
```

```
val _ = printRes' "system6" "AgenFindall" system7

(* 8: --AFindall-> *)
val BRsystem7 = makeBR system7
val mts7 = M.matches { agent = BRsystem7 , rule = Afindall }
val match7 = LazyList.lznth mts7 0
val system8 = Re.react match7
val _ = print "\n"
val _ = printRes' "system7" "Afindall" system8
val _ = print "\n"


(* output in the graphical SVG format *)
(* bgval2svg
val _ = outputsvgdoc "system1.svg" system1
val _ = outputsvgdoc_v
*)
(* bgbdnf2svg
val _ = outputsvgdoc_b
*)

(* print to file
open TextIO;

val os = openOut("matches.out")

fun printmatches ms =
    if LazyList.lznull(ms) then "Done\n"
    else ( output(os, (M.toString (LazyList.lzhd ms)) ^ "\n")
 ; printmatches (LazyList.lztl ms) )

val out_a = printmatches mts0

val _ = flushOut(os)
val _ = closeOut(os)
*)
```

A.2.2 Tour guide

```
(*
 Implementation of an approximation of the context-aware
 tourist guide GUIDE; see the MobiCom 2000 paper "Experiences
 of Developing and Deploying a Context-Aware Tourist Guide:
 The GUIDE Project" by K. Cheverst, N. Davies, K. Mitchell,
 and A. Friday.

 Limitation: No messaging.

 Attractions are from the virtual tour at
 http://www.lancasterukonline.net/visitors/v-tour/index.htm.

 A demo of the "real" application can be found at
 http://www.guide.lancs.ac.uk/whatisguide.html.

 Ebbe Elsborg, December 2006
*)

(* use "l.sml"; *)
open TextIO;
open List;

(* Basic entities *)
datatype attraction =
  Att of string (* name *)
       * string (* info *)
       * string (* more info *)

val voidLoc = "voidLoc"
val voidAtt = Att("voidAtt","","")
fun deattract (Att tup) = tup

type link = string
type device = link
type location = link
type display = string
type btnid = string
type btnval = string
```

```
type message = string
type tourflag = bool
type path = (location * attraction) list
type group = device list
type connector = bool

type state = display * location * attraction list
           * message list * (btnid * btnval) list * tourflag
           * path * location * attraction * connector

datatype event =
         DeviceObserved of device * location
       | DeviceLost of device
       | ButtonClicked of btnid

type Queue = event list
type Stack = state list

(* Attractions *)
val castle = Att("Lancaster Castle","info","more-info")
val williamson = Att("Williamson Park and the Ashton
                      Memorial",
                     "info","more-info")
val queenvic = Att("Queen Victoria monument","info",
                   "more-info")
val seagull = Att("Saltayre Seagull Colony","info",
                  "more-info")
val halfmoon = Att("Half Moon Bay","info","more-info")
val market = Att("Farmers Street Market","info","more-info")
val canal = Att("Lancaster Canal","info","more-info")
val nightingale = Att("Lancashire Witches","info",
                      "more-info")
val spooky = Att("Spooky Paths","info","more-info")
val ruxton = Att("Dr. Buck Ruxton's House","info",
                 "more-info")
val priory = Att("Lancaster Priory","info","more-info")
val merchants = Att("J Atkinson & Co, Tea & Coffee Merchants",
                    "info","more-info")
val humbug = Att("Humbugs Sweetshop","info","more-info")
val cemetery = Att("Lancaster Cemetery","info","more-info")
```

```
val museum = Att("City Museum and The King's Own Museum",
                 "info","more-info")
val cottage = Att("Cottage Museum + Roman Bath-House","info",
                  "more-info")
val meetinghouse = Att("Friends Meeting House","info",
                       "more-info")
val theatre = Att("The Grand Theatre","info","more-info")
val judges = Att("Judges Lodgings + Doll Museum","info",
                 "more-info")
val maritime = Att("Maritime Museum","info","more-info")
val leisure = Att("Lancaster Leisure Park","info",
                  "more-info")
val millenium = Att("Lancaster Millenium Bridge and
                     River Lune Millennium Park","info",
                     "more-info")
val music = Att("The Music Room","info","more-info")
val stpeters = Att("St Peters RC Cathedral","info",
                   "more-info")
val townhall = Att("Lancaster Town Hall","info","more-info")

(* Popular attractions and location-attraction
   relationship *)

val popular : attraction list =
  [castle,williamson,queenvic,seagull,halfmoon,market,canal,
   nightingale,spooky,ruxton,priory]

val locAtts : (location * attraction list) list =
    [("TIC", []),
     ("l1", [castle,Att("'dummy'","info","moreinfo")]),
     ("l2", [williamson]),
     ("l3", [queenvic]),
     ("l4", [seagull]),
     ("l5", [halfmoon]),
     ("l6", [market]),
     ("l7", [canal]),
     ("l8", [nightingale]),
     ("l9", [spooky]),
     ("l10", [ruxton]),
     ("l11", [priory])]
```

```
(* Constants *)
val our_id : device ref = ref "ab:cd:ef:gh:ij:kl"
val our_grp : group ref = ref ["d1","d2","d3"]

(* Auxiliary functions *)
fun app f [] = ()
  | app f (x::xs) = ( f x ; app f xs )

fun exists p [] = false
  | exists p (x::xs) = p x orelse exists p xs

fun filter p [] = []
  | filter p (x::xs) = if p x then x :: filter p xs
                       else filter p xs

fun last x = (hd o rev) x

fun lookup1 map x =
    let fun loop [] = NONE
          | loop ((l,a)::m) = if l = x then SOME l else loop m
    in loop map end

fun lookup2 map x =
    let fun loop [] = NONE
          | loop ((l,a)::m) = if l = x then SOME a else loop m
    in loop map end

fun findLoc map a =
    let fun loop [] = NONE
          | loop ((l,a')::m) =
            if exists (fn x => a = x) a'
            then SOME l
            else loop m
    in loop map end

fun getAtts l = case lookup2 locAtts l
                  of NONE => []
                   | SOME x => x
```

```
fun first [] = []
  | first ((l,a)::m) = l :: first m

fun second [] = []
  | second ((l,a)::m) = a @ second m

fun remElm e [] = []
  | remElm e (x::xs) = if e = x then xs else x :: remElm e xs

fun remDubs [] = []
  | remDubs (x::xs) =
    if exists (fn y => y = x) xs then remDubs xs
    else x :: remDubs xs

fun remBtn b [] = []
  | remBtn b ((i,n)::m) = if b = i then m
                          else (i,n) :: remBtn b m

fun remBtns x [] = []
  | remBtns [] x = x
  | remBtns (b::bs) x = remBtn b x @ remBtns bs x

(*
fun addBtn b l = if exists (fn x => x = b) l
                   then l else l @ [b]
*)

fun findNextElm [] x = NONE
          | findNextElm [e] x = SOME e
          | findNextElm (e::e'::m) x =
            if x = e then SOME e' else findNextElm (e'::m) x

fun findPrevElm [] x = NONE
  | findPrevElm (e::es) x = findNextElm (rev (e::es)) x

val locs = first locAtts
val atts = second locAtts

val std_btns = [("locator","Locator"),
                ("tour","Tour"),
```

```
                ("message","Message"),
                ("quit","Quit")]

val tour_btns = [("message","Message"),
                 ("back","Back"),
                 ("quit","Quit")]

fun whichLocBtns hl =
    case locs
     of [] => []
      | (l::ls) =>
        if ls = [] then [("select-loc","Select")]
        else
            if hl = l
            then [("select-loc","Select"),
                  ("next-loc","Next"),
                  ("back","Back"),
                  ("quit","Quit")]
            else
                if hl = last ls
                then [("select-loc","Select"),
                      ("previous-loc","Previous"),
                      ("back","Back"),
                      ("quit","Quit")]
                else [("select-loc","Select"),
                      ("previous-loc","Previous"),
                      ("next-loc","Next"),
                      ("back","Back"),
                      ("quit","Quit")]

fun whichSelBtns [] ha = []
  | whichSelBtns [x] ha = [("select-att","Select")]
  | whichSelBtns (x::xs) ha =
    if ha = x then [("select-att","Select"),
                    ("next-tour","Next")]
    else
        if ha = last xs
        then [("select-att","Select"),
              ("previous-tour","Previous")]
        else [("select-att","Select"),
```

```
              ("previous-tour","Previous"),
              ("next-tour","Next")]

fun whichInfoBtns [] ha = []
  | whichInfoBtns [x] ha = [("info","Info")]
  | whichInfoBtns (x::xs) ha =
    if ha = x then [("info","Info"),
                    ("next-att","Next")]
    else
        if ha = last xs
        then [("info","Info"),
              ("previous-att","Previous")]
        else [("info","Info"),
              ("previous-att","Previous"),
              ("next-att","Next")]

fun whichTourBtns [] = []
  | whichTourBtns [b] = [("end","End")]
  | whichTourBtns (b::bs) = [("cont","Continue")]

(* State stack with operations *)
val stack : Stack ref = ref []

fun stackSize s =
  case s of [] => 0
          | (x::xs) => 1 + stackSize xs

fun push s = stack := s::(!stack)

fun pop () =
    case (!stack)
     of [] => NONE
      | (e::es) => let val _ = stack := es in SOME(e) end

(* Event queue with operations, 'enq' is visible from L *)
val queue : Queue ref = ref []

fun enq e = queue := (!queue)@[e]

fun deq () =
```

```
    case (!queue)
     of [] => NONE
      | (e::es) => let val _ = queue := es in SOME(e) end

(* gui *)
fun printDisp d = print("GUIDE: " ^ d ^ "\n")

fun printLocs (locs,hl) =
    let fun printLocList [] x = print "\n"
          | printLocList (l::ls) x =
            if x=l then ( print("*" ^ l ^ "  ")
                        ; printLocList ls x )
            else ( print(l ^ "  ") ; printLocList ls x )
    in ( print "Locations: " ; printLocList locs hl ) end

fun printAtts (alist,ha) =
    let fun printAttList [] att = print "\n"
          | printAttList (Att(n,i,m)::t)
                         (att as Att(x,i',m')) =
            if x=n then ( print("*" ^ n ^ "  ")
                        ; printAttList t att )
            else ( print(n ^ "  ") ; printAttList t att )
    in ( print "Attractions: " ; printAttList alist ha ) end

fun printCurLoc l =
    print("Location: Device " ^ !our_id ^
          " is in " ^ l ^ ".\n")

fun printBtns b =
    ( print "Buttons: ";
      app (print o (fn s => s ^ "   ") o #2) b;
      print "\n" )

fun printMsgs m =
    ( print "Messages: ";
      if m = [] then print "No messages."
      else app (print o (fn s => s ^ " ")) m;
      print "\n" )

fun printPath p =
```

```
    let (*fun printAttList [] = ()
          | printAttList (Att(n,i,m)::t) =
            ( print(" " ^ n) ; printAttList t )*)
        fun printPathList [] flag = print "\n"
          | printPathList ((l,Att(n,i,m))::t) flag =
            if flag
            then ( print("#" ^ "(" ^ l ^ ",");
                   print(" " ^ n); (*printAttList a;*)
                   print ")";
                   printPathList t false )
            else ( print(" -> " ^ "(" ^ l ^ ",");
                   print(" " ^ n); (*printAttList a;*)
                   print ")";
                   printPathList t false )
    in ( print "Path: " ; printPathList p true ) end

fun printCon c = if c then print "Connector: Connected.\n"
                 else print "Connector: Not connected.\n"

(* temp code begin *)
fun printStackSize s =
    print("stacksize: " ^ Int.toString(stackSize s) ^ "\n")

fun printStack s =
    let fun printTour flag = if flag then print "t: true\n"
                             else print "t: false\n"
        fun printElm (d,l,a,m,b,t,p,hl,ha,c) =
            ( printDisp(d);
              printCurLoc(l);
              printAtts(a,ha);
              printMsgs(m);
              printCon(c);
              printBtns(b);
              printTour(t);
              printPath(p);
              print("hl: " ^ hl ^ "\n");
              print("ha: " ^ #1(deattract(ha)) ^ "\n\n") )
        fun pip s = case s
                     of [] => print "\n"
                      | (e::es) => ( printElm e ; pip es )
```

```
    in ( print "#####\n"; pip s ; print "#####\n" ) end
(* temp code end *)

(* maybe do something intelligent later, e.g. find
   all-pairs shortest path, but this requires a parent map to
   be retrieved from the location model *)
fun calcPath path = path

fun locShow () =
    ( print "\n-----------------------------
               -----------------------------\n";
      case pop()
       of NONE => print "locShow(): empty stack\n"
        | SOME(d,l,a,m,b,t,p,hl,ha,c) =>
          ( printDisp(d);
            printLocs(locs,hl);
            printCurLoc(l);
            printCon(c);
            printBtns(b);
            push(d,l,a,m,b,t,p,hl,ha,c) );
          print "\nPlease press a button.\n";
          print "-----------------------------
                 -----------------------------\n" )

fun tourShow () =
    ( print "\n-----------------------------
               -----------------------------\n";
      case pop()
       of NONE => print "tourShow(): empty stack\n"
        | SOME(d,l,a,m,b,t,p,hl,ha,c) =>
          ( printDisp(d);
            printCurLoc(l);
            printAtts(a,ha);
            printPath(p);
            printMsgs(m);
            printCon(c);
            printBtns(b);
           push(d,l,a,m,b,t,p,hl,ha,c) );
          print "\nPlease press a button.\n";
          print "-----------------------------
```

```
                ----------------------------\n" )

fun guiShow () =
    ( print "\n------------------------------
              ----------------------------\n";
      case pop()
       of NONE => print "guiShow(): empty stack\n"
        | SOME(d,l,a,m,b,t,p,hl,ha,c) =>
          ( printDisp(d);
            printCurLoc(l);
            printAtts(a,ha);
            if t then printPath(p) else ();
            printMsgs(m);
            printCon(c);
            printBtns(b);
            push(d,l,a,m,b,t,p,hl,ha,c) );
          print "\nPlease press a button.\n";
          print "------------------------------
                 ----------------------------\n" )

(* Location events *)
fun deviceObserved loc (d,l,a,m,b,t,p,hl,ha,c) =
( print("deviceObserved(" ^ loc ^ ")\n");
    ( if loc = l then push(d,l,a,m,b,t,p,hl,ha,c)
      else let val new_atts = getAtts loc
               val (a',ha') = if new_atts = []
                              then ([],voidAtt)
                              else (new_atts,hd new_atts)
               val tmp_b = remBtn "back"
                                  ([("locator","Locator")]
                                  @ tour_btns)
               val tmp_info = (whichInfoBtns a' ha')
               val pred = fn (x,y) => not(x = loc)
               val p' = if t
                        then (calcPath o filter) pred p
                        else p
               val b' = if t then
                            if length p = 1 then
                                tmp_b @ tmp_info
                                @ [("end","End")]
```

```
                            else tmp_b @ tmp_info
                                 @ [("cont","Continue")]
                        else std_btns @ tmp_info
           in push(d,loc,a',m,b',t,p',hl,ha',c) end );
    guiShow() ; true
)

fun deviceLost (d,l,a,m,b,t,p,hl,ha,c) =
( print "deviceLost()\n";
  deviceObserved "an unknown location"
                 (d,l,a,m,b,t,p,hl,ha,c)
)

(* Button events *)
fun quitClicked (d,l,a,m,b,t,p,hl,ha,c) = false

fun backClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "backClicked()\n";
    ( guiShow() ; true )
)

fun infoClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "infoClicked()\n";
    let val d' = (#1(deattract ha)) ^ "\n"
                  ^ (#2(deattract ha))
        val b' = std_btns @ [("more-info","More info"),
                             ("back","Back")]
    in ( push(d,l,a,m,b,t,p,hl,ha,c);
         push(d',l,a,m,b',t,p,hl,ha,c);
         guiShow();
         true )
    end
)

fun moreInfoClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "moreInfoClicked()\n";
    let val d' = (#1(deattract ha)) ^ "\n"
                  ^ (#3(deattract ha))
        val b' = std_btns @ [("back","Back")]
    in ( push(d,l,a,m,b,t,p,hl,ha,c);
```

```
          push(d',l,a,m,b',t,p,hl,ha,c);
          guiShow();
          true )
    end
)

fun locatorClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "locatorClicked()\n";
    let val d' = "Please select one of the following
                  locations:\n"
        val b' = whichLocBtns hl
    in ( push(d,l,a,m,b,t,p,hl,ha,c);
         push(d',l,a,m,b',t,p,hl,ha,c);
         locShow();
         true )
    end
)

fun messageClicked (d,l,a,m,b,t,p,hl,ha,c) =
    ( print "Messaging is not implemented.\n";
      push(d,l,a,m,b,t,p,hl,ha,c);
      guiShow();
      true )

fun selectLocClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "selectLocClicked()\n";
  case pop()
   of NONE => ( print "empty stack\n" ; false )
    | SOME(d',l',a',m',b',t',p',hl',ha',c') =>
      let val new_atts = getAtts hl
          val (a'',ha'') = if new_atts = [] then ([],voidAtt)
                           else (new_atts,hd new_atts)
          val tmp_b = remBtn "back"
                             ([("locator","Locator")]
                             @ tour_btns)
          val tmp_info = (whichInfoBtns a'' ha'')
          val pred = fn (x,y) => not(x = hl)
          val p'' = if t' then (calcPath o filter) pred p'
                          else p'
          val b'' = if t' then
```

```
                        if length p'' = 1
                        then tmp_b @ tmp_info
                             @ [("end","End")]
                        else tmp_b @ tmp_info
                             @ [("cont","Continue")]
                    else std_btns @ tmp_info
      in ( push(d',hl,a'',m',b'',t',p'',hl,ha'',c);
           guiShow();
           true )
      end
)

fun previousLocClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "previousLocClicked()\n";
    let val hl' = case findPrevElm locs hl
                  of NONE => voidLoc | SOME loc => loc
        val b' = whichLocBtns hl'
    in ( push(d,l,a,m,b',t,p,hl',ha,c) ; locShow() ; true )
    end
)

fun nextLocClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "nextLocClicked()\n";
    let val hl' = case findNextElm locs hl
                  of NONE => voidLoc | SOME loc => loc
        val b' = whichLocBtns hl'
    in ( push(d,l,a,m,b',t,p,hl',ha,c) ; locShow() ; true )
    end
)

fun previousTourAttClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "previousTourAttClicked()\n";
    let val ha' = case findPrevElm a ha
                  of NONE => voidAtt | SOME att => att
        val b' = remDubs (tour_btns @ [("pop","Popular")]
                          @ (whichSelBtns a ha')
                          @ [("done","Done")])
    in ( push(d,l,a,m,b',t,p,hl,ha',c) ; tourShow() ; true )
    end
)
```

```
fun previousAttClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "previousAttClicked()\n";
    let val ha' = case findPrevElm a ha
                   of NONE => voidAtt | SOME att => att
        val tmp_b = std_btns @ (whichInfoBtns a ha')
        val b' = if t
                 then tmp_b @ (whichTourBtns p)
                 else tmp_b
    in ( push(d,l,a,m,b',t,p,hl,ha',c) ; guiShow() ; true )
    end
)

fun nextTourAttClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "nextTourAttClicked()\n";
    let val ha' = case findNextElm a ha
                   of NONE => voidAtt | SOME att => att
        val b' = remDubs (tour_btns @ [("pop","Popular")]
                          @ (whichSelBtns a ha')
                          @ [("done","Done")])
    in ( push(d,l,a,m,b',t,p,hl,ha',c) ; tourShow() ; true )
    end
)

fun nextAttClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "nextAttClicked()\n";
    let val ha' = case findNextElm a ha
                   of NONE => voidAtt | SOME att => att
        val tmp_b = std_btns @ (whichInfoBtns a ha')
        val b' = if t
                 then tmp_b @ (whichTourBtns p)
                 else tmp_b
    in ( push(d,l,a,m,b',t,p,hl,ha',c) ; guiShow() ; true )
    end
)

fun tourClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "tourClicked()\n";
    let val d' = "Please select a subset of the following
                  attractions, one at a time.\n"
```

```
        val a' = atts
        val t' = true
        val p' = []
        val ha' = case a' of [] => voidAtt | (x::xs) => x
        val b' = tour_btns @ [("pop","Popular")]
                           @ (whichSelBtns a' ha')
    in ( push(d,l,a,m,b,t,p,hl,ha,c);
         push(d',l,a',m,b',t',p',hl,ha',c);
         tourShow();
         true )
    end
)

fun popularClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "popularClicked()\n";
    let val d' = "You chose the popular tour. Please proceed
                  to the tour location marked with '#'.\n"
        val new_atts = getAtts l
        val (a',ha') = if new_atts = [] then ([],voidAtt)
                       else (new_atts,hd new_atts)
        val b' = std_btns @ (whichInfoBtns a' ha')
                 @ [("cont","Continue")]
        val getLocs =
            fn att => case findLoc locAtts att
                       of NONE => (voidLoc,att)
                        | SOME loc => (loc,att)
        val locatts = map getLocs popular
        val pred = fn (x,y) => not(x = voidLoc)
        val p' = (calcPath o filter) pred locatts
    in ( push(d',l,a',m,b',t,p',hl,ha',c);
         tourShow();
         true )
    end
)

fun selectAttClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "selectAttClicked()\n";
    let val a' = filter (fn x => not(x = ha)) a
        val p' = case findLoc locAtts ha
                  of NONE => []
```

```
                    | SOME loc => calcPath (p @ [(loc,ha)])
        val ha' = case findPrevElm a' ha
                  of NONE => voidAtt | SOME att => att
        val tmp_b = tour_btns @ [("pop","Popular")]
                    @ (whichSelBtns a' ha')
                    @ [("done","Done")]
        val b' = remDubs tmp_b
    in ( push(d,l,a',m,b',t,p',hl,ha',c);
         tourShow();
         true )
    end
)

fun doneClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "doneClicked()\n";
  case pop()
   of NONE => ( print "empty stack\n" ; false )
    | SOME(d',l',a',m',b',t',p',hl',ha',c') =>
      let val d'' = "You have confirmed your tour.
                     Please proceed to the tour location
                     marked with '#'.\n"
          val b'' = if length p = 1 then b' @ [("end","End")]
                    else b' @ [("cont","Continue")] (* p>1 *)
      in ( push(d',l',a',m',b',t',p',hl',ha',c');
           push(d'',l,a',m,b'',t,p,hl,ha',c);
           tourShow();
           true )
      end
)

fun continueClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "continueClicked()\n";
  let val d' = "Please proceed to the next tour location,
                marked with '#'.\n"
      val b' = if length p = 2
               then (remBtn "cont" b) @ [("end","End")]
               else b (* p>2 *)
      val p' = tl p (* non-empty by invariant *)
  in ( push(d',l,a,m,b',t,p',hl,ha,c);
       tourShow();
```

```
       true )
  end
)

fun endClicked (d,l,a,m,b,t,p,hl,ha,c) =
( print "endClicked()\n";
  case pop()
   of NONE => ( print "empty stack\n" ; false )
    | SOME(d',l',a',m',b',t',p',hl',ha',c') =>
      let val (a'',ha'') =
              case getAtts l
                of [] => ([],voidAtt)
                 | (x::xs) => (x::xs,x)
          val b'' = std_btns @ (whichInfoBtns a ha)
          val t'' = false
      in ( push(d',l,a'',m',b'',t'',p',hl',ha'',c');
           guiShow();
           true )
      end
)

val button_clicks =
    [("quit", quitClicked),
     ("back", backClicked),
     ("info", infoClicked),
     ("more-info", moreInfoClicked),
     ("locator", locatorClicked),
     ("message", messageClicked),
     ("tour", tourClicked),
     ("select-loc", selectLocClicked),
     ("previous-loc", previousLocClicked),
     ("next-loc", nextLocClicked),
     ("previous-att", previousAttClicked),
     ("next-att", nextAttClicked),
     ("previous-tour", previousTourAttClicked),
     ("next-tour", nextTourAttClicked),
     ("select-att", selectAttClicked),
     ("done", doneClicked),
     ("pop", popularClicked),
     ("cont", continueClicked),
```

```
     ("end", endClicked)]

fun handleEvent event s =
    case event of
        DeviceObserved(dev,loc) =>
        if dev = !our_id then deviceObserved loc s
        else (push s ; true ) (* not about us, ignore *)
      | DeviceLost(dev) =>
        if dev = !our_id then deviceLost s
        else (push s ; true) (* not about us, ignore *)
      | ButtonClicked(bid) =>
        ( case lookup2 button_clicks bid
           of NONE => ( push s ; true ) (* unknown button,
                                           ignore *)
            | SOME f => f s )
      (* delete this?
      | ShortestPath(from,to,path) =>
          if from = !current_location then displayPath path
          else (* perhaps do something intelligent when we
                  are on the path, for now just ignore *)
               true *)

(* I/O *)
fun errmsg () =
    ( output(stdOut, "Unavailable button.
                      Please try again.\n");
      flushOut stdOut )

fun peel s =
    if String.size s <= 0 then ""
    else String.substring(s, 0, (String.size s)-1)

fun read () =
(*( printStackSize(!stack); printStack(!stack);*)
    case pop()
     of NONE => print "read(): empty stack\n"
      | SOME(s) =>
        ( let val (d,l,a,m,b,t,p,hl,ha,c) = s
          in case lookup1 b (peel (inputLine stdIn)) of
                 NONE => ( errmsg() ; push(s) ; read() )
```

```
               | SOME(btn) =>
                   ( enq(ButtonClicked btn);
                     if not (btn = "quit") andalso
                        handleEvent(ButtonClicked btn) s
                     then read()
                     else () )
          end )
(*)*)

(* Event loop *)
fun eventLoop () =
    case deq()
     of NONE => eventLoop()
      | SOME e => case pop()
                  of NONE =>
                       print "eventLoop(): empty stack\n"
                   | SOME s =>
                     if handleEvent e s
                     then
                       let val _ = inputLine stdIn
                       in eventLoop() end
                     else () (* halt *)

(* this function must be supplied by L *)
fun whereIs d = "dummy_location"

val init_events = [ButtonClicked("tour"),
                   ButtonClicked("select-att"),
                   ButtonClicked("next-tour"),
                   ButtonClicked("locator"),
                   ButtonClicked("next-loc"),
                   ButtonClicked("previous-loc"),
                   ButtonClicked("back"),
                   ButtonClicked("next-tour"),
                   ButtonClicked("select-att"),
                   ButtonClicked("next-tour"),
                   ButtonClicked("select-att"),
                   ButtonClicked("done"),
                   DeviceObserved("ab:cd:ef:gh:ij:kl","l1"),
                   ButtonClicked("info"),
```

```
                  ButtonClicked("more-info"),
                  ButtonClicked("back"),
                  ButtonClicked("back"),
                  DeviceObserved("ab:cd:ef:gh:ij:kl","l8"),
                  ButtonClicked("locator"),
                  ButtonClicked("next-loc"),
                  ButtonClicked("next-loc"),
                  ButtonClicked("select-loc"),
                  ButtonClicked("tour"),
                  ButtonClicked("pop"),
                  DeviceLost("ab:cd:ef:gh:ij:kl"),
                  ButtonClicked("quit")]

(* test button events or location events *)
val BTN_TEST = false

fun main () =
    ( let val disp = "Welcome to Lancaster\n"
          val loc = case first locAtts
                      of [] => ""
                       | (l::ls) => l
          val atts = getAtts loc
          val msgs = []
          val btns = std_btns
          val on_tour = false
          val tour_path = []
          val hilite_loc = loc
          val hilite_att = case atts
                             of [] => voidAtt
                              | (x::xs) => x
          val con = true
      in ( push(disp,loc,atts,msgs,btns,on_tour,
                tour_path,hilite_loc,hilite_att,con);
           queue := init_events;
           guiShow();
           if BTN_TEST then read() else eventLoop()
         )
      end )
```

A.3 π-calculus

This appendix contains standard π-calculus definitions and an i/o-type system, for easy reference.

Definition A.33 (Binding)**.** *In each of $a(x).P$ and $\nu x\, P$, the displayed occurrence of x is* binding *with* scope *P. An occurrence of a name in a process is* bound *it it is, or it lies within the scope of, a binding occurrence of the name. An occurrence of a name in a process is* free *it it is not bound.*

Definition A.34 (Substitution)**.** *A* substitution *is a function on names that is the identity except on a finite set.*

Notation A.35 (Substitution on names)**.** *Use σ to range over substitutions, and write σx for σ applied to x. The* support *of σ, $\mathsf{supp}(\sigma)$, is $\{x \mid \sigma x \neq x\}$, and the* co-support *of σ, $\mathsf{cosupp}(\sigma)$, is $\{\sigma x \mid x \in \mathsf{supp}(\sigma)\}$. Write $\mathsf{n}(\sigma)$ for the set of* names *of σ, which is $\mathsf{supp}(\sigma) \cup \mathsf{cosupp}(\sigma)$. Write $\{^{y_1,\ldots,y_n}/_{x_1,\ldots,x_n}\}$ for the substitution σ such that $\sigma x_i = y_i$ for each $i \in \{1,\ldots,n\}$ and $\sigma x = x$ for $x \notin \{x_1,\ldots,x_n\}$. If X is a set of names, write σX for $\{\sigma x \mid x \in X\}$.*

Definition A.36 (α-convertibility)**.**

1. *If the name x does not occur in the process P, then $\{^x/_y\}P$ is the process obtained by replacing each free occurrence of y in P by x.*

2. *A* change of bound names *in a process P is the replacement of a subterm $a(x).Q$ of P by $a(y).\{^y/_x\}Q$, or the replacement of a subterm $\nu x\, Q$ of P by $\nu y\, \{^y/_x\}Q$, where in each case y does not occur in Q.*

3. *Processes P and Q are α-*convertible*, $P =_\alpha Q$, if Q can be obtained from P by a finite number of changes of bound names.*

Convention A.37. *When considering a collection of processes and substitutions, it is assumed that the bound names of the processes are chosen to be different from their free names and from the names of the substitutions.*

Definition A.38 (Substitution on prefixes)**.** *The effect of applying a substitution σ to a prefix π is to replace each occurrence of each name x in π by σx.*

Definition A.39 (Substitution on processes)**.** *The process σP, obtained by*

applying σ to P is defined as follows, avoiding capture of names by binders:

$$\begin{aligned} \sigma(\pi.P) &\stackrel{\text{def}}{=} \sigma\pi.\sigma P \\ \sigma(P \mid Q) &\stackrel{\text{def}}{=} \sigma P \mid \sigma Q \\ \sigma(\nu x\, P) &\stackrel{\text{def}}{=} \nu x\, (\sigma P) \\ \sigma\mathbf{0} &\stackrel{\text{def}}{=} \mathbf{0}\,. \end{aligned}$$

Notation A.40 (Operator precedence)**.** *When writing processes as linear expressions parentheses are used to resolve ambiguity, and observe the conventions that prefixing and restriction bind more tightly than parallel composition. Further, substitutions bind more tightly than process operators. Sometimes parentheses are inserted merely to aid reading.*

Definition A.41 (Process context)**.** *A* process context *is a process term in which exactly one process subterm has been left out leaving a "hole" represented with notation* $[\cdot]$*. For a context C write* $C[P]$ *for the process resulting from "plugging" the process P into the hole of C, where the hole in C must occur in a position such that* $C[P]$ *is well-formed for an arbitrary process term P.*

Definition A.42 (Process congruence)**.** *An equivalence relation* $\mathcal{R}$ *on processes is a* process congruence *if* $(P, Q) \in \mathcal{R}$ *implies* $(C[P], C[Q]) \in \mathcal{R}$ *for every process context C.*

$$\textit{Processes}: \quad \frac{}{\Gamma \vdash \mathbf{0} : \diamond} \qquad \frac{\Gamma, x : L \vdash P : \diamond}{\Gamma \vdash (\nu x : L)\, P : \diamond} \qquad \frac{\Gamma \vdash P : \diamond \quad \Gamma \vdash Q : \diamond}{\Gamma \vdash P \mid Q : \diamond}$$

$$\frac{\Gamma \vdash a : \mathsf{i}S \quad \Gamma, y : S \vdash P : \diamond}{\Gamma \vdash a(y).P : \diamond} \qquad \frac{\Gamma \vdash a : \mathsf{o}T \quad \Gamma \vdash x : T \quad \Gamma \vdash P : \diamond}{\Gamma \vdash \bar{a}x.P : \diamond}$$

$$\textit{Subtyping}: \quad \frac{}{T \leq T} \qquad \frac{S \leq S' \quad S' \leq T}{S \leq T} \qquad \frac{}{\#T \leq \mathsf{i}T} \qquad \frac{}{\#T \leq \mathsf{o}T}$$

$$\frac{S \leq T}{\mathsf{i}S \leq \mathsf{i}T} \qquad \frac{T \leq S}{\mathsf{o}S \leq \mathsf{o}T} \qquad \frac{T \leq S \quad S \leq T}{\#S \leq \#T}$$

$$\textit{Names}: \quad \frac{}{\Gamma, x : T \vdash x : T} \qquad \frac{\Gamma \vdash x : S \quad S \leq T}{\Gamma \vdash x : T}$$

Table A.1: i/o-type rules for sfπ.

Bibliography

[ACH+01] Mike Addlesee, Rupert W. Curwen, Steve Hodges, Joe Newman, Pete Steggles, Andy Ward, and Andy Hopper. Implementing a Sentient Computing System. *IEEE Computer*, 34(8):50–56, August 2001.

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid Automata: An algorithmic approach to the specification and verification of hybrid systems. *Hybrid Systems*, 736:209–229, 1993. LNCS.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AEH+04] Jesper Andersen, Ebbe Elsborg, Fritz Henglein, Jakob Grue Simonsen, and Christian Stefansen. Compositional Specification of Commercial Contracts. In *Preliminary Proceedings of the 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA'04)*, pages 103–110. University of Cyprus Report TR-2004-6, 2004.

[AEH+06] Jesper Andersen, Ebbe Elsborg, Fritz Henglein, Jakob Grue Simonsen, and Christian Stefansen. Compositional Specification of Commercial Contracts. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):485–516, November 2006. Special Section on Leveraging Applications of Formal Methods.

[BBD+06] Mikkel Bundgaard, Lars Birkedal, Søren Debois, Ebbe Elsborg, Arne J. Glenstrup, Thomas Hildebrandt, Troels C. Damgaard, Robin Milner, and Henning Niss. Bigraphical

Programming Languages for Pervasive Computing. In Thomas Strang, Vinny Cahill, and Aaron Quigley, editors, *Pervasive 2006 Workshop Proceedings – The 1st International Workshop on Combining Theory and Systems Building in Pervasive Computing (CTSB'06)*, pages 653–658, 2006. Position paper.

[BBR02] Martin Bauer, Christian Becker, and Kurt Rothermel. Location Models from the Perspective of Context-Aware Applications and Mobile Ad Hoc Networks. *Personal and Ubiquitous Computing*, 6(5-6):322–328, December 2002.

[BCC01] Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Boxed ambients. In *Proceedings of TACS'01*, volume 2215 of *LNCS*, pages 38–63. Springer-Verlag, 2001.

[BD05] Christian Becker and Frank Dürr. On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, 9:20–31, January 2005. Springer-Verlag.

[BDE+05] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Hildebrandt, and Henning Niss. Bigraphical Models of Context-aware Systems. Technical Report 74, The IT University of Copenhagen, November 2005.

[BDE+06] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Hildebrandt, and Henning Niss. Bigraphical Models of Context-aware Systems. In Luca Aceto and Anna Ingólfsdóttir, editors, *Proceedings of FoSSaCS'06*, volume 3921 of *LNCS*, pages 187–201. Springer-Verlag, 2006.

[BDGM06] Lars Birkedal, Troels C. Damgaard, Arne J. Glenstrup, and Robin Milner. Matching of bigraphs. Technical Report ITU-TR-2006-88, The IT University of Copenhagen, June 2006.

[BDH06] Lars Birkedal, Søren Debois, and Thomas Hildebrandt. Sortings for Reactive Systems. In Christel Baier and Holger Hermanns, editors, *Proceedings of CONCUR'06*, volume 4137 of *LNCS*, pages 248–262. Springer-Verlag, 2006.

[BDH08] Lars Birkedal, Søren Debois, and Thomas Hildebrandt. On the Construction of Sorted Reactive Systems. In *Proceedings of CONCUR'08*, LNCS, pages 218–232. Springer-Verlag, 2008.

[BDNN98] Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Control flow analysis for the π-calculus. In Davide Sangiorgi and Robert De Simone, editors, *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 84–98. Springer-Verlag, 1998.

[Ber03] Martin Berger. An interview with Robin Milner. `http://www.dcs.qmul.ac.uk/~martinb/interviews/milner/`, September 2003. Cambridge.

[BH06] Mikkel Bundgaard and Thomas Hildebrandt. Bigraphical Semantics of Higher-Order Mobile Embedded Resources with Local Names. In Arend Rensink, Reiko Heckel, and Barbara König, editors, *Proceedings of GT-VC'05*, volume 154 of *ENTCS*, pages 7–29. Elsevier, 2006.

[BP04] Pietro Braione and Gian Pietro Picco. On Calculi for Context-Aware Coordination. In *Proceedings of COORDINATION'04*, volume 2949 of *LNCS*, pages 38–54. Springer-Verlag, 2004.

[Bra03] Pietro Braione. *On Calculi for Context-Aware Systems*. PhD thesis, Politecnico di Milano, Dipartimento di Electronica e Informazione, 2003.

[BS01] Barry Brumitt and Steven Shafer. Topological World Modeling Using Semantic Spaces. In *UbiComp 2001: Workshop on Location Modeling for Ubiquitous Computing*, 2001.

[BS06] Mikkel Bundgaard and Vladimiro Sassone. Typed polyadic pi-calculus in bigraphs. In *Proceedings of PPDP'06*, pages 1–12. ACM Press, 2006.

[Bun07] Mikkel Bundgaard. *Semantics of Higher-Order Mobile Embedded Resources and Local Names*. PhD thesis, The IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S, August 2007.

[BZD02] Michael Beigl, Tobias Zimmer, and Christian Decker. A location model for communicating and processing of context. *Personal and Ubiquitous Computing*, 6(5-6):341–357, December 2002.

[CCK+05] Dan Chalmers, Jon Crowcroft, Marta Kwiatkowska, Robin Milner, Vladimiro Sassone, and Morris Sloman. Global Ubiquitous Computing: Design and Science. Final draft, a newer version of the document can be found at `http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/manifesto.pdf`, June 2005.

[CDMF00] Keith Cheverst, Nigel Davies, Keith Mitchell, and Adrian Friday. Experiences of developing and deploying a context-aware tourist guide: the GUIDE project. In *Proceedings of MobiCom'00*, pages 20–31, 2000.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, June 2000.

[Cha88] K. Mani Chandy. Parallel Program Design: A Foundation. Addison-Wesley, 1988. ISBN: 0-201-05866-9.

[CK00] Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Department of Computer Science, Dartmouth College, 2000.

[CMS05] Giovanni Conforti, Damiano Macedonio, and Vladimiro Sassone. Spatial Logics for Bigraphs. In *Proceedings of ICALP'05*, volume 3580 of *LNCS*, pages 766–778. Springer-Verlag, 2005.

[D'A99] Pedro R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. PhD thesis, Department of Computer Science, University of Twente, November 1999.

[DA00] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *Human Factors in Computing Systems (CHI'00): Workshop on The What, Who, Where, When, and How of Context-Awareness*, 2000.

[Dam08] Troels C. Damgaard. *Developing Bigraphical Languages*. PhD thesis, The IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S, December 2008. Preprint.

[DB96] Pedro R. D'Argenio and Ed Brinksma. A calculus for Timed Automata. In *Proceedings of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1135 of *LNCS*, pages 110–129. Springer-Verlag, 1996.

[DB05] Troels C. Damgaard and Lars Birkedal. Axiomatizing Binding Bigraphs (revised). Technical Report TR-2005-71, The IT University of Copenhagen, October 2005.

[DB06] Troels C. Damgaard and Lars Birkedal. Axiomatizing Binding Bigraphs. *Nordic Journal of Computing*, 13(1-2):58–77, June 2006.

[DD05] Søren Debois and Troels C. Damgaard. Bigraphs by example. Technical Report TR-2005-61, The IT University of Copenhagen, March 2005.

[Deb08] Søren Debois. *Sortings and Bigraphs*. PhD thesis, The IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S, April 2008.

[Dom01] Svetlana Domnitcheva. Location modeling: State of the Art and Challenges. In Michael Beigl, Phil Gray, and Daniel Salber, editors, *UbiComp'01: Proceedings of the Workshop on Location Modeling for Ubiquitous Computing*, pages 13–19, 2001.

[DR03] Frank Dürr and Kurt Rothermel. On a Location Model for Fine-Grained Geocast. In Anind K. Dey, Albrecht Schmidt, and Joseph F. McCarthy, editors, *Proceedings of UbiComp'03*, LNCS, pages 18–35. Springer-Verlag, 2003.

[DRD+00] Alan Dix, Tom Rodden, Nigel Davies, Jonathan Trevor, Adrian Friday, and Kevin Palfreyman. Exploiting Space and Location as a Design Framework for Interactive Mobile Systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(3):285–321, September 2000.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science (EATCS Series). Springer-Verlag, first edition, 2006. ISBN: 978-3-540-31187-4.

[EHS08] Ebbe Elsborg, Thomas Hildebrandt, and Davide Sangiorgi. Type Systems for Bigraphs. Technical Report 110, The IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark, October 2008.

[EHS09] Ebbe Elsborg, Thomas Hildebrandt, and Davide Sangiorgi. Type Systems for Bigraphs. In Christos Kaklamanis and Flemming Nielson, editors, *Proceedings of TGC'08*, LNCS. Springer-Verlag, 2009. To appear.

[Els06] Ebbe Elsborg. Bigraphical Location Models. Technical Report 94, The IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S, September 2006.

[FcRH04] Chen-Liang Fok, Gruia catalin Roman, and Gregory Hackmann. A Lightweight Coordination Middleware for Mobile Computing. In Rocco De Nicola, Gian Luigi Ferrari, and Greg Meredith, editors, *Proceedings of COORDINATION'04*, volume 2949 of *LNCS*, pages 135–151. Springer-Verlag, 2004.

[FF97] Matthias Felleisen and Daniel P. Friedman. *The Little MLer*. The MIT Press, December 1997.

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A Calculus of Mobile Agents. In *Proceedings of CONCUR'96*, volume 1119 of *LNCS*, pages 406–421. Springer-Verlag, 1996.

[GG06] Jens Chr. Godskesen and Olena Gryn. Modelling and verification of security protocols for ad hoc networks using uppaal. In *Proceedings of the 18th Nordic Workshop on Programming Theory (NWPT'06)*, 2006.

[GHK08] Jens Chr. Godskesen, Hans Hüttel, and Morten Kühnrich. Verification of correspondence assertions in a calculus for mobile ad hoc networks. In *Proceedings of FOCLASA'08*, 2008.

[GM07a] Davide Grohmann and Marino Miculan. Directed Bigraphs. In *Proceedings of MFPS XXIII*, volume 173 of *ENTCS*, pages 121–137. Elsevier, 2007.

[GM07b] Davide Grohmann and Marino Miculan. Reactive Systems over Directed Bigraphs. In Luís Caires and Vasco T. Vasconcelos, editors, *Proceedings of CONCUR'07*, volume 4703 of *LNCS*, pages 380–394, 2007.

[GM08a] Davide Grohmann and Marino Miculan. An Algebra for Directed Bigraphs. In Ian Mackie and Detlef Plump, editors, *Proceedings of TERMGRAPH'07*, volume 203 of *ENTCS*, pages 49–63. Elsevier, 2008.

[GM08b] Davide Grohmann and Marino Miculan. Controlling resource access in Directed Bigraphs. In Juan de Lara Claudia Ermel and Reiko Heckel, editors, *Proceedings of GT-VMT'08*, volume 10 of *Electronic Communications of the EASST*. European Association of Software Science and Technology, 2008.

[God06] Jens Chr. Godskesen. Formal verification of the ARAN protocol using the applied Pi-calculus. In *Proceedings of Sixth International IFIP WG 1.7 Workshop on Issuses in the Theory of Security, (WITS)*, pages 99–113, 2006.

[God07] Jens Chr. Godskesen. A calculus for mobile ad hoc networks. In Amy L. Murphy and Jan Vitek, editors, *Proceedings of COORDINATION'07*, volume 4467 of *LNCS*, pages 132–150. Springer-Verlag, 2007.

[God08] Jens Chr. Godskesen. A calculus for mobile ad-hoc networks with static location binding. In *Proceedings of EXPRESS'08*, 2008.

[Gor08] Daniele Gorla. Towards a Unified Approach to Encodability and Separation Results for Process Calculi. In Franck van Breugel and Marsha Chechik, editors, *Proceedings of CONCUR'08*, number 5201 in LNCS, pages 492–507. Springer-Verlag, 2008.

[GV08] Orna Grumberg and Helmut Veith, editors. *25 Years of Model Checking*, volume 5000 of *LNCS*. Springer-Verlag, 2008. ISBN: 978-3-540-69849-4.

[Har00] Robert Harper. Type Systems for Programming Languages (DRAFT). Notes, Spring 2000. `http://www.cs.cmu.edu/~rwh/misc/tspl.pdf`.

[HB01] Jeffrey Hightower and Gaetano Borriello. A Survey and Taxonomy of Location Systems for Ubiquitous Computing.

Technical Report UW-CSE 01-08-03, University of Washington, August 2001.

[HBB02] Jeffrey Hightower, Barry Brumitt, and Gaetano Borriello. The location stack: a layered model for location in ubiquitous computing. In *Proceedings of the 4th IEEE workshop on mobile computing systems and applications (WMCSA'02)*, pages 22–28, June 2002.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of LICS'96*, pages 278–292. IEEE Computer Society Press, 1996.

[Hen04] Matthew Hennessy. Context-awareness: Models and Analysis. Talk at 2nd UK-UbiNet Workshop, slides at `www.cogs.susx.ac.uk/users/matthewh/talks.html`, May 2004.

[Hen05] Matthew Hennessy. Towards a calculus for nominal mobile agents. Talk at TGC'05, slides at `http://www.cogs.susx.ac.uk/users/matthewh/talks/tgc05.pdf`, April 2005.

[Hen08] Matthew Hennessy. *Distributed Pi-Calculus*. Cambridge University Press, 2008. ISBN-13: 9780521873307.

[HHS+02] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The Anatomy of a Context-Aware Application. *Wireless Networks*, 8:187–197, February 2002.

[HIR02] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling Context Information in Pervasive Computing Systems. In *Proceedings of PERVASIVE'02*, volume 2414 of *LNCS*, pages 167–180. Springer-Verlag, 2002.

[HNO06] Thomas Hildebrandt, Henning Niss, and Martin Olsen. Formalising Business Process Execution with Bigraphs and Reactive XML. In Paolo Ciancarini and Herbert Wiklicky, editors, *Proceedings COORDINATION'06*, volume 4038 of *LNCS*, pages 113–129. Springer-Verlag, 2006.

[HNOW05] Thomas Hildebrandt, Henning Niss, Martin Olsen, and Jacob Winther. Distributed Reactive XML. In *Proceedings of the 1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord'05)*, 2005.

[Hop00] Andy Hopper. Sentient Computing? *Phil. Trans. R. Soc. Lond., A*, 358:2349–2358, August 2000. An abridged and updated version of the Royal Society Clifford Paterson Lecture 1999.

[IK04] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the Pi-calculus. *Theoretical Computer Science*, 311(1-3):121–163, January 2004.

[Jen07] Ole Høgh Jensen. *Mobile Processes in Bigraphs (Draft)*. PhD thesis, University of Cambridge, 2007. Submitted.

[JM03] Ole Høgh Jensen and Robin Milner. Bigraphs and Transitions. In *Proceedings of POPL'03*, pages 38–49. ACM Press, 2003.

[JM04] Ole Høgh Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge – Computer Laboratory, February 2004.

[JPR04] Christine Julien, Jamie Payton, and Gruia-Catalin Roman. Reasoning About Context-Awareness in the Presence of Mobility. In Antonio Brogi, Jean-Marie Jacquet, and Ernesto Pimentel, editors, *Proceedings of FOCLASA'04*, volume 97 of *ENTCS*, pages 259–276, 2004.

[JR02] Christine Julien and Gruia-Catalin Roman. Egocentric context-aware programming in ad hoc mobile environments. In *Proceedings of the 10th International Symposium in the Foundations of Software Engineering*, pages 21–30, 2002.

[JR06] Christine Julien and Gruia-Catalin Roman. Egospaces: Facilitating Rapid Development of Context-Aware Mobile Applications. *IEEE Transactions on Software Engineering*, 32(5):281–298, May 2006.

[JS02] Changhao Jiang and Peter Steenkiste. A Hybrid Location Model with a Computable Location Identifier for Ubiquitous Computing. In *Proceedings of UbiComp'02*, pages 246–263. Springer-Verlag, 2002.

[KÖ5] Barbara König. A General Framework for Types in Graph Rewriting. *Acta Informatica*, 42(4):349–388, December 2005. Special issue: Types in concurrency, Part II.

[KBP06a] Mikkel Baun Kjærgaard and Jonathan Bunde-Pedersen. A Formal Model for Context-Awareness. Technical Report RS-06-2, BRICS, February 2006.

[KBP06b] Mikkel Baun Kjærgaard and Jonathan Bunde-Pedersen. Towards a Formal Model of Context-Awareness. In Thomas Strang, Vinny Cahill, and Aaron Quigley, editors, *Proceedings of CTSB'06*, 2006. Position paper.

[KMS04] Marta Kwiatkowska, Robin Milner, and Vladimiro Sassone. Science for global ubiquitous computing. Bulletin of the EATCS, 2004. Pages 325-333, volume 82.

[KMT08] Jean Krivine, Robin Milner, and Angelo Troina. Stochastic Bigraphs. In *Proceedings of 24th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXIV)*, volume 218 of *ENTCS*, pages 73–96. Elsevier, 2008.

[Kob02] Naoki Kobayashi. A type system for lock-free processes. *Information and Computation*, 177:122–159, September 2002. Issue 2.

[Kob06] Naoki Kobayashi. A new type system for deadlock-free processes. In *Proceedings of CONCUR'06*, volume 4137 of *LNCS*, pages 233–247. Springer-Verlag, 2006.

[Lei01] James Judi Leifer. *Operational Congruences for Reactive Systems*. PhD thesis, University of Cambridge Computer Laboratory and Trinity College, September 2001. Technical Report 521.

[Leo98] Ulf Leonhardt. *Supporting Location-Awareness in Open Distributed Systems*. PhD thesis, Department of Computing, University of London, May 1998.

[LM00a] James J. Leifer and Robin Milner. Deriving Bisimulation Congruences for Reactive Systems. In Catuscia Palamidessi, editor, *Proceedings of CONCUR'00*, LNCS, pages 243–258. Springer-Verlag, 2000.

[LM00b] James Judi Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In *Proceedings of CONCUR'00*, volume 1877 of *LNCS*, pages 243–258. Springer-Verlag, 2000.

[LM06] James J. Leifer and Robin Milner. Transition systems, link graphs, and Petri nets. *Mathematical Structures in Computer Science*, 16(6):989–1047, December 2006.

[LS05] Stephen Lack and Paweł Sobociński. Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications*, 39(2):522–546, 2005.

[MB97] John McCarthy and Saša Buvač. Formalizing Context. In Atocha Aliseda, Rob van Glabbeek, and Dag Westerståhl, editors, *Computing Natural Language: Working papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language*, pages 99–135. Stanford University, 1997.

[Mil91] Robin Milner. The Polyadic π-Calculus: a Tutorial. Technical Report ECS-LFCS-91-180, Computer Science Department, University of Edinburgh, October 1991. Published in F. L. Hamer, W. Brauer and H. Schwichtenberg, editors, Logic and Algebra of Specification. Springer-Verlag, 1993.

[Mil96] Robin Milner. Calculi for interaction. *Acta Informatica*, 33(8):707–737, 1996.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999. ISBN: 0-521-65869.

[Mil02] Robin Milner. Computing in Space, 1 May 2002. A lecture by Robin Milner, for the opening of the Computer Laboratory's William Gates Building at the University of Cambridge.

[Mil04a] Robin Milner. Axioms for bigraphical structure. Technical Report UCAM-CL-TR-581, University of Cambridge – Computer Laboratory, February 2004.

[Mil04b] Robin Milner. Bigraphs for Petri Nets. In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, volume 3098 of *LNCS*, pages 686–701. Springer-Verlag, July 2004.

[Mil04c] Robin Milner. Bigraphs whose names have multiple locality. Technical Report UCAM-CL-TR-603, University of Cambridge – Computer Laboratory, September 2004.

[Mil05a] Robin Milner. Axioms for bigraphical structure. *Mathematical Structures in Computer Science*, 15(6):1005–1032, December 2005.

[Mil05b] Robin Milner. Bigraphs: A Tutorial. Slides, April 2005. `http://www.cl.cam.ac.uk/users/rm135/bigraphs-tutorial.pdf`.

[Mil06a] Robin Milner. Pure bigraphs: Structure and dynamics. *Information and Computation*, 204(1):60–122, January 2006.

[Mil06b] Robin Milner. Ubiquitous computing: Shall we understand it? *The Computer*, pages 383–389, July 2006. Issue 4.

[Mil07] Robin Milner. Local Bigraphs and Confluence: Two Conjectures. *ENTCS*, 175(3), June 2007.

[Mil09] Robin Milner. *From semantics to Computer Science; Essays in Memory of Gilles Kahn*, chapter The tower of informatic models. Cambridge University Press, 2009. To appear.

[MP04] Amy L. Murphy and Gian Pietro Picco. Using Coordination Middleware for Location-Aware Computing: A Lime Case Study. In Rocco De Nicola, Gian Luigi Ferrari, and Greg Meredith, editors, *Proceedings of COORDINATION'04*, volume 2949 of *LNCS*, pages 263–278. Springer-Verlag, 2004.

[MPR06] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. LIME: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents. *ACM Transactions on Software Engineering (TOSEM)*, pages 1–48, 2006.

[MS97] Irwin Meisels and Mark Saaltink. The Z/EVES reference manual (for version 1.5). Technical Report TR-97-5493-03d, ORA Canada, September 1997. `http://www.ift.ulaval.ca/~jodesharnais/glo21941/ZEves/ZEvesRefMan.pdf`.

[NGP05] Rocco De Nicola, Daniele Gorla, and Rosario Pugliese. Basic Observables for a Calculus for Global Computing. In *Proceedings of ICALP'05*, volume 3580 of *LNCS*, pages 1226–1238. Springer-Verlag, 2005.

[NH04] Sebastian Nanz and Chris Hankin. Static analysis of routing protocols for ad-hoc networks. In *Proceedings of the 2004 ACM SIGPLAN and IFIP WG 1.7 Workshop on Issues in the Theory of Security (WITS'04)*, pages 141–152, 2004.

[NH06] Sebastian Nanz and Chris Hankin. Formal security analysis for ad-hoc networks. In *Proceedings of the 2004 Workshop on Views on Designing Complex Architectures (VODCA'04)*, volume 142 of *ENTCS*, pages 195–213, 2006.

[OJDA01] Thomas O'Connell, Peter Jensen, Anind K. Dey, and Gregory D. Abowd. Location in the aware home. In Michael Beigl, Phil Gray, and Daniel Salber, editors, *UbiComp'01: Location Modeling for Ubiquitous Computing*, 2001.

[PlaBC] Plato. The republic, book vii, 360 B.C. Translation by Benjamin Jowett.

[Pra00] Salil Pradhan. Semantic Location. *Personal and Ubiquitous Computing*, 4(4):213–216, 2000.

[PS96] Benjamin C. Pierce and Davide Sangiorgi. Typing and Subtyping for Mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

[PT00] Benjamin C. Pierce and David N. Turner. Pict: A Programming Language Based on the Pi-Calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 455–494. MIT Press, 2000.

[RCS06] Vinny Reynolds, Vinny Cahill, and Aline Senart. Requirements for an ubiquitous computing simulation and emulation environment. In *Proceedings of InterSense'06*, volume 138 of *ACM International Conference Proceeding Series*, page Article No. 1. ACM Press, 2006.

[Rep99] John H. Reppy. *Concurrent Programming in ML*. Cambridge University Press, 1999. ISBN: 0-521-48089-2.

[RJP04] Gruia-Catalin Roman, Christine Julien, and Jamie Payton. A Formal Treatment of Context-Awareness. In *Proceedings of FASE'04*, volume 2984 of *LNCS*, pages 12–36, 2004.

[RLU94] Mike Rizzo, Peter F. Linington, and Ian Utting. Integration of location services in the open distributed office. Technical Report 14-94*, University of Kent, Computing Laboratory, University of Kent, August 1994.

[RM02] Gruia-Catalin Roman and Peter J. McCann. A Notation and Logic for Mobile Computing. *Formal Methods in System Design*, 20(1):47–68, January 2002.

[RMP97] Gruia-Catalin Roman, Peter J. McCann, and Jerome Y. Plun. Mobile UNITY: Reasoning and Specification in Mobile Computing. *ACM Transactions on Software Engineering Methodology*, 6(3):250–282, July 1997.

[Rot03] Jörg Roth. Flexible positioning for location-based services. *IADIS International Journal on WWW/Internet*, 1(2):18–32, 2003.

[SAW94] Bill Schilit, Norman Adams, and Roy Want. Context-Aware Computing Applications. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, 1994.

[SBG99] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to Context than Location. *Computers & Graphics Journal*, 23(6):893–902, December 1999.

[SC02] Kumaresan Sanmugalingam and George Coulouris. A Generic Location Event Simulator. In Gaetano Borriello and Lars Erik Holmquist, editors, *Proceedings of UbiComp'02*, volume 2498 of *LNCS*, pages 308–315. Springer-Verlag, 2002.

[Sch95] Bill N. Schilit. *A Context-Aware System Architecture for Mobile Distributed Computing*. PhD thesis, Columbia University, May 1995.

[SLP04] Thomas Strang and Claudia Linnhoff-Popien. A context modelling survey. In *UbiComp'04: First International Workshop on Advanced Context Modelling, Reasoning And Management*, 2004.

[SS03] Vladimiro Sassone and Paweł Sobociński. Deriving bisimulation congruences: 2-categories vs. precategories. In *Proceedings of FOSSACS'03*, volume 2620 of *LNCS*, pages 409–424, 2003.

[SS05a] Vladimiro Sassone and Paweł Sobociński. Locating reaction with 2-categories. *Theoretical Computer Science*, 333(1-2):297–327, March 2005.

[SS05b] Vladimiro Sassone and Paweł Sobociński. Reactive systems over cospans. In *Proceedings of LICS'05*, pages 311–320. IEEE Computer Society Press, 2005.

[ST94] Bill Schilit and Marvin Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5):22–32, September/October 1994.

[SW01] Davide Sangiorgi and David Walker. *The Pi-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001. ISBN: 0-521-78177-9.

[Ter06] Sotirios Terzis. Combining Theory and Systems Building – Experiences and Challenges. In Thomas Strang, Vinny Cahill, and Aaron Quigley, editors, *Proceedings of CTSB'06*, 2006. Position paper.

[WBB06] Torben Weis, Christian Becker, and Alexander Brändle. Towards a programming paradigm for pervasive applications based on the ambient calculus. In *Proceedings of CTSB'06*, May 2006. Position paper.

[Wei91] Mark Weiser. The Computer for the 21st Century. In *Scientific American Ubicomp Paper after Scientific American edititing*, volume 265, pages 94–104. Scientific American, September 1991. Issue 3.

[Wei93] Mark Weiser. Hot Topics – Ubiquitous Computing. *IEEE Computer*, 26(10):71–72, October 1993.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcao, and Jon Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.

[WJH97] Andy Ward, Alan Jones, and Andy Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, 1997.

[Zim05] Pascal Zimmer. A Calculus for Context-Awareness. Technical Report RS-05-27, BRICS, August 2005.