Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis

Spatial Logics for Semistructured Resources

Giovanni Conforti

Referee
Luca Cardelli

Referee
Silvano Dal-Zilio

Supervisor
Giorgio Ghelli

Chair
Andrea Maggiolo-Schettini

September 23, 2005

Abstract

Spatial Logics have been recently proposed as modal logics inspecting the 'spatial' nature of models (as opposed to 'temporal logics' inspecting model behavior). Spatial Logics, essentially, lift constructors (and structural properties) of underlying models to the logical level, obtaining new 'spatial' logical connectives. The semantics of spatial logics is model dependant: different properties in the model turn into different spatial connectives. The main aim of this thesis is an in-depth understanding of Spatial Logics, in particular about the notion of separation and abstraction in different models and their influence in the decidability of the logic.

The new meta-model of *bigraphs* (proposed by Milner recently) is more general and more extensible than the models studied for spatial logics so far. We propose a general meta-logical framework, BiLog, inspired by the bigraphical structure. BiLog is contextual and parametric wrt structure and congruence of the model. This framework is interesting for spatial logics comparison and is very promising being a primary step for a truly general logic for distributed calculi with semistructured resources.

We instantiate BiLog to describe bigraphs and their components and we show that the resulting logics naturally embed Spatial Logics previously proposed in literature.

We also study how the introduction of name abstraction in the model and of quantifiers in the logic influences the decidability problem in Spatial Logics. Finally, we hint how Spatial Logics can be applied to model, describe and reason about a particular kind of semistructured resource: Web data.

To my father who taught me to listen; without listening silently and with attention the others I wouldn't have learnt nothing.
To my mother who taught me several things; in particular she taught me to learn and, most importantly, to do it without pauses, without ever stopping.
To my brother Michele who taught me to 'work hard' and to take my responsibilities.
To my brother Nicola who taught me to communicate and not to fear.
To my brother Bruno who taught me to think and to create from myself.
But above the others, to my Daniela who taught me to love.
To all the others 'mine' in Sicily and in Sardinia; you also taught me many things.
Thanks to all of you for supporting me and let me fly.
With the hope that someday I will teach all things I learnt.

A mio padre che mi ha insegnato ad ascoltare; senza ascoltare silenziosamente e attentamente gli altri non avrei appreso nulla.
A mia madre che mi ha insegnato innumerevoli cose; in particolare mi ha insegnato ad imparare e, ancor meglio, di farlo senza soste, senza mai accontentarsi.
A mio fratello Michele che mi ha insegnato a 'faticare' e a prendermi delle responsabilitá.
A mio fratello Nicola che mi ha insegnato a comunicare e a non aver paura.
A mio fratello Bruno che mi ha insegnato a pensare e creare per mio conto.
Ma soprattutto, alla mia Daniela che mi ha insegnato ad amare.
A tutti gli altri 'miei' in Sicilia e Sardegna; anche voi mi avete insegnato tanto.
Grazie a voi tutti che mi avete supportato e mi avete lasciato volare.
Con la speranza che tutto ció che ho imparato lo possa un giorno insegnare.

Contents

List of Tables

List of Figures

Acknowledgements

A significative part of this Thesis has been produced during a *Marie Curie* visit to Informatics and Cognitive Science Department at University of Sussex. I thank *DiSco* and *MIKADO* european projects for financing this period abroad.

During this three years of Ph.D. I have apprecciated the company and discussions with Carlo Sartiani, Dario Colazzo, Paolo Manghi, Massimo Bartoletti, Ivan Lanese in Pisa and with Philippe Bidinger, Rohit Chadra, Federico Cozzi, Matthew Hennessy, Bernhard Reus, Jan Schwingammer, Adrian Francalanza, Pawel Sobocinski and all other memebers of the theory lab of Sussex University.

I had some discussion that have influenced this work in some way with Peter O'Hearn, Robin Milner and Thomas Hildebrandt.

I want to thank in particular three persons. *Vladimiro Sassone*, my supervisor in Sussex University, has been foundamental for my 'initiation' to the bigraph theory and for the last orientation of this Thesis. I want to thank him also for the numerous rich discussions about work and life we had. *Giorgio Ghelli*, my supervisor in Pisa, has introduced me to spatial logics and semistructured data. His contribution was decisive for the decidability (and undecidability) results of this Thesis. And last but not least, *Damiano Macedonio* for the precious collaboration, the hospitality, and the infinite patience he demonstrated. Many points of this thesis are the result of days (and nights) of work together with him.

A last thank to Daniela for having supported me in any occasion.

Introduction

In the last years, with the consolidation of World Wide Web, we have seen a large and increasing interest and research effort in two, among many, computer science areas

- formal models and calculi for distributed and mobile computation. These models (and corresponding semantics, type systems and logics) were mainly studied as a theoretical foundation for concurrent, distributed (and mobile) programming. In particular many calculi have been proposed and studied to describe mobile computation: among the others π-calculus [93] and mobile ambients [46]. Recently *Bigraphs* [80] have been proposed as a truly general (meta)model for global systems. They appear to encompass several existing calculi, including π-calculus [80], ambient calculus [81], and petri-nets [95].

- models, languages and tools for describing, querying and manipulating *semi-structured data*, i.e. data with an irregular and unstable structure. After an intensive investigation and combined effort by document and database communities, *XML* (eXtensive Markup Language) and many XML-related languages (e.g. XPath, XQuery, XSLT) are becoming the standard languages for semi-structured data.

The similarities between structured process calculi (in particular mobile ambients [47]) and semistructured data have been addressed rather recently by Cardelli in [37]. In particular, the models share the nested labelled tree structure and the aim to be distributed and "global". These similarities are becoming even more important with the advent of "global computing". We now have process calculi that make use of tree-structured data (or more generally tree-structured resources): as messages (e.g. in [18, 8]), or as a distributed data repository (e.g. in [71]). In addition, we have many applications needing a formal model able to describe both distributed mobile processes and semi-structured resources (e.g. Web Service Orchestration, protocols for Peer2Peer Systems, ActiveXML, and the Microsoft Cω).

However, the two aforementioned models have also some differences. A big first difference is in the order of (parallel) composition. Putting two processes in parallel is an unordered operation, while the composition of two trees in XML is usually order-preserving. However, if we interpret semistructured data as representing entities (e.g. in databases or data integration), or as distributed P2P database where

order is not definable, the unordered model is preferred. Thus, the unordered tree model is adequate for semistructured data also and actually a query language for semistructured data (TQL) based on the Ambient Logic was proposed [41] and implemented [54].

Spatial Logics, i.e. logics featuring operators inspired by the model structure, have been proposed recently to describe both processes on one side [28, 45] and semistructured data on the other [32, 38, 40, 33]. A natural question arise: "is there a general theory behind these spatial logics?", and in particular "can we use the similarities of the model to build a general meta-model for all structures and then build a logic on this meta-model?". The main aim of this thesis is to find an answer to these questions. Our idea is that a meta-model is already being studied by Milner et al. for process calculi and consists in *bigraphs* (and bigraphical reactive systems). Bigraphs seem very general and mirror easily the structure of semistructured data, actually they could be proposed as model on their own (as we do in the fourth part of the Thesis). Thus, the idea to develop a logic for bigraphs seems very appealing. On the other side we have decidability and complexity of the logic we are using. At the beginning of this thesis many questions about decidability of spatial logics were open. We closed some of them and we tried to construct a spatial meta-logic without using operators we know would lead to undecidability.

Thesis contribution The main contribution of this thesis are:

- We introduce a spatial meta-logical framework for resources inspired by bigraphs. This framework can be instantiated to model bigraphical components and embed spatial logics. This corresponds to the second part of this Thesis. An extended abstract on this contribution is published in [61].

- We study the decidability problem in spatial logics on abstract trees with hidden names. Hidden name quantification can be decomposed in revelation and fresh name quantification. We obtain a surprising result, while spatial tree logic with fresh name quantification is a rich decidable logic, the introduction of revelation in a very simple logic leads to undecidability. This corresponds to the third part of this Thesis. An extended abstract of this work is published in [57].

- We propose bigraphs as a model for Web Data and BiLog as a logic to describe them, this idea is published in [60] and presented in the fourth part of the thesis.

Structure of the dissertation This dissertation is divided in four parts. The first part is an introduction on spatial logic (Chapter 1) and bigraphs (Chapter 2). The second part describes our meta-logical framework (Chapter 3) and the interesting logics obtained as its instances (Chapter 4). The third part introduces the

model of trees with abstract names (Chapter 5), studies the decidability of freshness (Chapter 6) and the undecidability of revelation (Chapter 7). Finally, in part four we give an overview on Web Data described with Spatial Logics (Chapter 8, Chapter 9) and we model Web Data with bigraphs (Chapter 10).

Part I

Background: Spatial Logics and Bigraphs

Chapter 1

Spatial Logics Overview

In this chapter we present the state of the art in the emerging field of *spatial logics*, that is logics featuring operators inspired by the (spatial) *structure* of the model. There have been many proposals and applications in this field despite the topic is very recent. Up to now, models for spatial logics include computational structures such as heaps [106, 101], trees [37], trees with hidden names [38], graphs [40], concurrent objects [30] as well as process calculi such as the π-calculus [28, 29] and the Ambient Calculus [43, 45].

In particular two main research streams have emerged, motivated by different applications:

- A stream is centered on Hoare-style assertion languages to describe (and verify) properties of programs that manipulate structured data.
- The other stream is centered on model-checking processes (or data structures) w.r.t. logical operators mirroring the underlying structure of the model. Some models can evolve during the time and in this case the logic usually includes also temporal connectives to describe the model behaviour. Thus, this kind of spatial logics are *intensional* in the sense that they can observe how the model structure evolves during a computation.

Actually, the term *Spatial Logic* is referred in the literature mainly to logics in the second stream, while the first stream is usually associated to the term *Separation Logic* (the name of the first proposal in that direction). If we consider the structural component of models only (no transitions), we can refer to both kinds of logics as *Static Spatial Logics*, because both make use of connectives inspecting the 'spatial structure' of the model as opposed to the 'temporal' behaviour.

Even without temporal connectives, static spatial logics differ in some details, such as their different notions of names and the partiality/totality of constructors in the underlying models. Throughout this overview we will try to address the differences and similarities among these logics.

In the following chapter we will introduce bigraphs as a model able to unify and generalize the introduced static spatial logic models. Building on this model we will

propose in Part II a logical framework that aims to be a generalization of the static spatial logics presented in this chapter.

Dynamic models In this chapter we will also present briefly some dynamic models, i.e. models that can *compute* (evolve in time by little transitions). We will show how this behaviour is described in spatial logics with temporal connectives. These models will be briefly re-explored in the conclusion of the first part of the thesis by considering how the introduction of dynamics in the model corresponds to a definition of a bigraphical reactive system. We will then propose an idea for a generalization of the temporal connectives based on the 'reactive system definition'. In this context the bigraphical model becomes even more appealing as it is both a good generalization of spatial logic models and a meta-model for concurrent processes in general.

Name restriction/abstraction Spatial logics are mainly used to describe (semi-) structured nominal resources (i.e. labelled trees or graphs, processes communicating via named channels). In some cases names are hidden or protected in the model, that is the logic (or query language) cannot express them. To describe nominal resources without having names for them can be difficult, for this task Cardelli and Gordon in [44] propose 'spatial' logical operators that make use of placeholder names/variables in order to inspect the nominal resources without knowing the real values for protected names. We will introduce and study such kind of spatial logic operators, and their influence on the decidability of the logic, in Part III.

1.1 Separation Logic for Heaps

We start our overview of spatial logics from the logic with a 'simpler' (not hierarchical) model structure. Separation logic describes *heap* structures, that is collections of identified locations containing values. The identifier of a heap location is usually called its *address*.

A heap is a natural model of memory in dynamic programming. The identifiers for locations are stored in variables (usually in the stack) in order to refer to memory. In addition, memory values can refer to other memory locations using a *pointer* mechanism to represent data structures like lists and trees. Reasoning about programs that manipulate this kind of data structures is difficult because of the *sharing* that pointers induce. Here, sharing means that the same location can be referred from several points in memory. The sharing of pointers originates also a problem known as *aliasing*, that is the same resource could be modified using different pointers to that resource; this can become puzzling for programmers that confuse the creation/clonation of a resource with the creation of a pointer to that resource.

Many proposal have attempted to extend Hoare logic, or to define new logics dealing with shared mutable data structures. But local reasoning about heap content can result more complex than for the store, because heap locations are not identified by variables names. In particular, the possibility of sharing data makes the specification of even simple properties in the usual predicate logic difficult. The main reason of this is the so-called *frame problem*. Each time we want to specify a property of data structures involving separated locations of the heap, we must explicitly specify that these locations are not related (i.e. are *separated*). This approach results in a not scalable formalism. *Separation logic* is mainly proposed as a scalable formalism for describing properties of shared mutable data structures. The basic idea is to localize assertions by describing little pieces of state that can be composed by connectives that intrinsically constrain separation (i.e. separating conjunction).

In addition, the separating constraint can be also used to specify non-interference in concurrent programs. The basic idea is that, just as program variables are syntactically partitioned into groups owned by different processes and resources, so the heap should be similarly partitioned by separating conjunctions in the proof of the program. We now introduce the heap model, the logic and the applications of separation logic.

1.1.1 Heap Model

We introduce the heap model as a collection of binary memory cells. Heaps denote finite partial functions of the form $\mathbf{Addr} \rightharpoonup_{fin} \mathbf{Val} \times \mathbf{Val}$. They map a finite set of addresses (identifying the memory locations) to couples of values (the location content).

As is common for spatial logics, we define models as terms and we formulate equational properties of models as a structural congruence between terms. Terms are constructed as combinations of building blocks (single locations and the empty heap in this case). The natural combination of heaps is a partial operation, because heaps sharing addresses cannot be combined. Thus, the combination $h * h'$ implicitly constrains domains to be disjoint (we denote the disjointness with $h \# h'$). In this case the structure is essentially a partial commutative monoid. In terms of resources we have a heap resource that can be described locally as a composition of little heap pieces. These pieces are collections of the building blocks locations identified by their addresses.

Table 1.1.1. *Heap Terms (over* **Addr**, **Val***) and congruence*

$h, h' ::=$	heaps
emp	empty heap
$(i \mapsto v_1, v_2)$	location $i \in \mathbf{Addr}$, $v_1 \in \mathbf{Val}$, $v_2 \in \mathbf{Val}$
$h * h'$	disjoint combination with $h \# h'$, i.e. $dom(h) \cap dom(h') = \emptyset$

where $dom(emp) = \emptyset$, $dom((i \mapsto v_1, v_2)) = \{i\}$, $dom(h * h') = dom(h) \cup dom(h')$

$h \equiv h * emp$
$h * h' \equiv h' * h$
$h * (h' * h'') \equiv (h * h') * h''$

Usually we have $\mathbf{Addr} \subset \mathbf{Val}$, which corresponds to having expressible addresses (we can implement pointers). In addition $nil \notin \mathbf{Addr}$ is a particular value used to denote the pointer to nothing.

Remark 1.1.1. *When an arithmetic on addresses is definable (e.g. addresses are natural numbers), a common way of defining heaps is with unary memory locations $(i \mapsto v)$. In this case consecutively allocated heap cells can be expressed as follows*

$$(i \mapsto v_0, \ldots, v_n) \stackrel{def}{=} (i \mapsto v_0) * (i + 1 \mapsto v_1) * \ldots * (i + n \mapsto v_n)$$

Example 1.1.2. *The following heap implements a list of natural numbers with length four.*

$$(i_1 \mapsto 1, i_2) * (i_2 \mapsto 2, i_3) * (i_3 \mapsto 6, i_4) * (i_4 \mapsto 24, nil)$$

Notice that addresses i_j are expressed as values and are mutually different, otherwise the heap is not defined. In addition, if we swap the locations we obtain a congruent heap term representing the same structure. The particular value nil represents the end of the list.

Example 1.1.3. *The heap $(i_1 \mapsto a, i_2) * (i_2 \mapsto b, i_1)$ implements a two-elements circular linked list, with a and b in the data fields.*

1.1.2 Propositional Separation Logic

Separation logic is a spatial logic describing heap resources. As in heap terms we have the composition constructor, in the logic we define a new connective of *separating conjunction* $\phi * \psi$. This connective can be interpreted with respect to a heap h as: *h can be split into two separated sub-heaps h' and h'' such that ϕ is true in h' and ψ is true in h''*. This connective is spatial over the heap, i.e. its semantics depends on the structure of the heap wrt it is interpreted. Combining separating conjunction with basic assertion on the shape of simple (portions of) heaps (singleton and empty predicates) we obtain a formalism that can express *strictly exact* assertions, that is formulas describing exactly the content of the heap.

The basic idea of separating conjunction is implicit in early work of Burstall [27] and is strongly connected to the parallel composition of the Ambient Logic [43]. It was explicitly described by Reynolds in lectures in the fall of 1999; then an

intuitionistic logic based on this idea was described independently in [105] and in [78] (where the concept of separating implication was introduced).

The separating implication $\phi \mathrel{-\!\!*} \psi$, also called *magic wand*, is the separating conjunction adjunct and it can be interpreted wrt a heap h as: *Composing h with a heap satisfying ϕ we obtain a heap that satisfies ψ*. Note that composition can be performed iff heaps are separated, so this assertion says nothing about heaps with overlapping domains. Both connectives (conjunction and implication) are *multiplicative* like linear logic connectives, i.e. $\phi * \phi \neq \phi$. But we usually want to express also classical predicates that make use of additive conjunction and implication. Combining additive connectives with multiplicative ones it is possible to express not strictly exact predicates such as $(\phi \wedge \psi) * \mathbf{T}$, asserting that *the heap contains a sub-heap satisfying both ϕ and ψ* ($\mathbf{T}$ stands for "true").

The integration of additive connectives and multiplicative ones is studied in [104] where the logic of *bunched implications* is introduced. In this logic the two forms of implications (additive and multiplicative) coexist giving an interesting logic for resources. The separation logic can be viewed as an instance of the resource interpretation of the logic of bunched implications.

Separation logic formulas are interpreted w.r.t. a *state* of the system. A state is a pair (s, h) where s is an environment for variables (the *store*) and h is a heap. The store, sometime called stack, is a partial map from variables to values.

In Table 1.1.2 we report the propositional fragment of separation logic and we suppose, for simplicity, to have a set of values $\mathbf{Val} \stackrel{def}{=} \mathbf{Addr} \cup \{nil\}$.

Table 1.1.2. *Propositional Separation Logic*

$E, E' ::=$	Value Expressions
x, y	Variables
$\mathbf{nil}$	Nil expression
$\phi, \psi ::=$	Formulas
$E = E'$	Equality
$\mathbf{F}$	Falsity
$\phi \Rightarrow \psi$	Implication
$(E \mapsto E_1, E_2)$	Heap binary cell
$\mathbf{emp}$	Empty heap
$\phi * \psi$	Composition
$\phi \mathrel{-\!\!*} \psi$	Composition adjunct
Forcing relation with $[\![x]\!]_s \stackrel{def}{=} s(x)$, $[\![\mathbf{nil}]\!]_s \stackrel{def}{=} nil$	
$(s,h) \models E = E'$	$\stackrel{def}{=} [\![E]\!]_s = [\![E']\!]_s$
$(s,h) \models \mathbf{F}$	$\stackrel{def}{=}$ never
$(s,h) \models \phi \Rightarrow \psi$	$\stackrel{def}{=}$ if $(s,h) \models \phi$ then $(s,h) \models \psi$
$(s,h) \models (E \mapsto E_1, E_2)$	$\stackrel{def}{=} h \equiv ([\![E]\!]_s \mapsto [\![E_1]\!]_s, [\![E_2]\!]_s)$
$(s,h) \models \mathbf{emp}$	$\stackrel{def}{=} h \equiv emp$
$(s,h) \models \phi * \psi$	$\stackrel{def}{=} \exists h_1, h_2.\, h \equiv h_1 * h_2$ and $(s,h_1) \models \phi$ and $(s,h_2) \models \psi$

$$(s,h) \models \phi \mathrel{-\!\!*} \psi \qquad \stackrel{def}{=} \forall h'.\, (h * h')\downarrow \text{ and } (s,h') \models \phi \text{ implies } (s, h * h') \models \psi$$

Standard logical connectives ($\mathbf{T}$, $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$) are defined as derived operators, e.g. $\neg\phi \stackrel{def}{=} (\phi \Rightarrow \mathbf{F})$. The resulting assertion language allows us to express memory properties in a compact and clear way. An interesting derived connective is the *monotone* binary cell assertion

$$(E \hookrightarrow E_1, E_2) \stackrel{def}{=} (E \mapsto E_1, E_2) * \mathbf{T}$$

This connective is monotone in the sense that if it holds for a small portion of a heap then it holds also for any bigger portion. It states that the current heap contains a binary cell (rather than consist entirely of that cell).

Example 1.1.4. *The expression $(s,h) \models (x \mapsto y, z)$ describes the heap h in terms of the variables stored in s. In particular it expresses that h consists of a sole location identified by $s(x)$ storing the pair of values $(s(y), s(z))$. Now we observe how the binary cell assertion (and its monotone version) interact with the additive and multiplicative conjunctions:*

- *the formula $(x_1 \mapsto y, z) * (x_2 \mapsto y, z)$ describes a heap with exactly two binary cells with addresses x_1 and x_2 respectively. These cells contain the same values. Notice that this property can hold only for stacks s such that $s(x_1) \neq s(x_2)$;*
- *the formula $(x_1 \mapsto y, z) \wedge (x_2 \mapsto y, z)$ holds for heaps satisfying simultaneously both conjuncts. This means that the heap is a single binary cell and the store s is such that $s(x_1) = s(x_2)$.*
- *the formula $(x_1 \hookrightarrow y, z) * (x_2 \hookrightarrow y, z)$ describes a heap containing* at least *two separated binary cells with addresses x_1 and x_2 respectively. Also in this case the formula constrains $s(x_1) \neq s(x_2)$;*
- *the formula $(x_1 \hookrightarrow y, z) \wedge (x_2 \hookrightarrow y, z)$ constrains the heap to contain sub-heaps satisfying the conjuncts, but it impose no constraint that these sub-heaps are different (or separated). Thus, this property can hold when $s(x_1) = s(x_2)$.*

Example 1.1.5. *The following formula describes a two-element circular linked list.*

$$(x \mapsto E_1, y) * (y \mapsto E_2, x)$$

Heaps of the form of Example 1.1.3 satisfy this formula when the store s is such that $[\![x]\!]_s = i_1$, $[\![y]\!]_s = i_2$, $[\![E_1]\!]_s = a$, and $[\![E_2]\!]_s = b$.

Separating conjunction is adequate to describe structures with pointers in a compact and scalable way. On the other hand, the separating implication gives us the possibility to express structural conditional properties of heap data structures.

Example 1.1.6. *The formula* $(x \mapsto E_1, y) * ((x \mapsto E_2, y) \mathrel{-\!\!*} \psi)$ *states that* ψ *holds if we change the* x *cell content. More generally, if* ϕ_1 *and* ϕ_2 *are exact formulas, then the formula* $\phi_1 * (\phi_2 \mathrel{-\!\!*} \psi)$ *describes what happens when we substitute in the heap the sub-heap described by* ϕ_1 *with the sub-heap described by* ϕ_2*.*

In addition, the *magic wand* can express properties with implicit universal quantification on heap structures. To give an idea of the expressive power we can internalize the quantification over heaps in the logic: the statement $(s, emp) \models \mathbf{T} \mathrel{-\!\!*} \psi$ holds iff the formula ψ is satisfied by all heaps (with a fixed store s).

1.1.3 Separation Logic with Quantifiers

More interesting examples can be expressed in the separation logic with quantification over names. The interpretation of quantification is classical, but its interaction with the separation and the points-to relation is not trivial. Suppose to extend the propositional fragment with the existential quantification over values (that is the logic presented in [101]).

$$(s, h) \models \exists x.\, \phi \quad \stackrel{def}{=} \quad \exists a \in \mathbf{Val}.\, (s[x \mapsto a], h) \models \phi$$

In this logic we can express properties on the *structure* of the memory abstracting from the values contained in (or the addresses of) the memory cell.

Example 1.1.7. *We can state that the heap contains a two-elements list starting from the address stored in* z *as follows:*

$$\exists y.(z \mapsto x_1, y) * (y \mapsto x_2, \mathbf{nil}) * \mathbf{T}$$

The heap in Example 1.1.2 satisfies this formula when s *maps* z *to* i_3*,* x_1 *to 6, and* x_2 *to 24 (the last two elements of the list). Notice how the variable* y *is used to bind the two memory cells only. The quantification is on all values, but some values are implicitly excluded because of the separating conjunction (in this case* $z \neq y$*).*

We can generalize this notion using the following recursive specification of lists in Separation Logic:

$$\begin{array}{lcl} \mathbf{list}\ z\ \epsilon & \stackrel{def}{=} & z = \mathbf{nil} \\ \mathbf{list}\ z\ x : \vec{a} & \stackrel{def}{=} & \exists y.\, (z \mapsto x, y) * \mathbf{list}\ y\ \vec{a} \end{array}$$

This definition is scalable because we avoid explicit and verbose disequality constraint for all involved addresses. When $\{z \mapsto i_1, x_1 \mapsto 1, x_2 \mapsto 2, x_3 \mapsto 6, x_4 \mapsto 24\} \subset s$ *and* h *is the heap in Example 1.1.2 we have* $(s, h) \models \mathbf{list}\ z\ x_1x_2x_3x_4$

1.1.4 Reasoning with Separation Logic

The classic approach for proving properties of imperative programming was introduced by Tony Hoare and is based on the notions of *preconditions* and *postconditions*. In this approach the programs are depicted as transformers of states and the Hoare triple $\{\phi\}\ \mathbf{C}\ \{\psi\}$ expresses that every time we start to execute the command $\mathbf{C}$ in a state satisfying the precondition ϕ, the resulting state satisfies the postcondition ψ. Pre and postconditions are expressed in an *assertion* language describing the state, usually first order logic.

In an imperative programming language the classic simple instruction that modifies the state of the program is the *assignment* $x := e$, assigning to the variable x in the current state the value corresponding to the expression e. The semantics of this command can be described by the following triple:

$$\{true\}\ x := e\ \{x = e\}$$

meaning that whatever the initial state is (precondition always true), the resulting state after the assignment satisfies the equality $x = e$. Another use of Hoare triples is in the *backward reasoning*, in this case we are trying to find the *weakest precondition* for a triple whose resulting state satisfies a determined property (e.g. we know what we want at the end of the computation, but we don't know which is the precondition or the command to obtain it). The backward reasoning style of Hoare triple for assignment is

$$\{\phi[x \leftarrow e]\}\ x := e\ \{\phi\}.$$

where $\phi[x \leftarrow e]$ substitutes syntactically the occurrences of the variable x in the assertion ϕ with the expression e. When the underlying model comprises a heap with mutable data structures (i.e. with pointers), backward reasoning for commands modifying the heap is difficult to express using first order logic assertions.

In [101] the authors show how separation logic can be used as an assertion language to give axiomatic semantic to a low-level imperative programming language. Essentially, the storage model is based on heap and unrestricted address arithmetic, and the classical assignment instruction is extended in order to access and modify the heap. In particular the command $[x] := e$ modifies the heap content in the address stored in the variable x with the value corresponding to e. Notice that the command can *fail* when the corresponding address in the heap is not allocated.

The assignment command can be described by the following triple

$$\{(x \mapsto -)\}\ [x] := e\ \{(x \mapsto e)\}$$

and the backward reasoning version of the triple is simply stated as

$$\{(x \mapsto -) * ((x \mapsto e) \mathrel{-\!\!*} \phi)\}\ [x] := e\ \{\phi\}$$

Essentially, to have a final state satisfying ϕ there must be a location at address x (whose content will be lost) in the state preceding the assignment, and we know that

substituting that location content with e the resulting heap must satisfy ϕ. In the previous approach we had an implicit substitution $\phi[x \leftarrow e]$, while now we perform a substitution explicitly in the assertion $(x \mapsto -) * ((x \mapsto e) \mathrel{-\!\!*} \phi)$ that internalizes separation of resources.

In [101] *small local axioms* are proposed for assignment commands using Hoare triples and separation logic. The idea of local axioms is to refer to the area of heap accessed by the corresponding command only. The core of the axiomatic system is completed by structural rules for auxiliary variable elimination, variable substitution, consequence and the usual *rule of constancy* of Hoare logic

$$\frac{\{\phi\}\ \mathbf{C}\ \{\psi\}}{\{\phi \wedge H\}\ \mathbf{C}\ \{\psi \wedge H\}} \quad Modifies(\mathbf{C}) \cap Free(H) = \emptyset$$

is replaced by the following *Frame Rule*:

$$\frac{\{\phi\}\ \mathbf{C}\ \{\psi\}}{\{\phi * H\}\ \mathbf{C}\ \{\psi * H\}} \quad Modifies(\mathbf{C}) \cap Free(H) = \emptyset$$

This rule codifies the notion of local behaviour and is fundamental for generalizing local specifications. It requires the use of the separating conjunction in order to constraint not-interference between heaps. In [115] the frame rule soundness and completeness (in the sense that the systems does not need any other frame axiom) are proved.

The small axioms are simple but not practical. In [101] some structural rules are proposed to derive more convenient laws.

Reynolds' paper [106] is a survey on separation logic that includes some extensions and interesting examples of program specification in presence of shared mutable data structures. We refer here the reader to some separation logic applications

- specification of imperative programs manipulating shared data structures, in [106] same specification examples are given for the following structures: lists, doubly-linked lists, trees, dags (directed acyclic graphs) and heap-allocated arrays;
- prove program correctness wrt the provided specification, an interesting example is provided in [114] where an algorithm for marking structures that contain sharing and cycles is proved correct using separation logic reasoning;
- concurrent programs reasoning, the work [100] observes how separation logic can be used also for ownership transfer, e.g. for synchronization via counting semaphores.

1.2 Spatial Logics for Trees

We have seen how separation logic can describe properties of heaps (flat collections of addressed locations). Now we proceed the overview on spatial logics by observing

what happens if we describe a hierarchical model. The usual notion of hierarchy on resources is modeled by an *unordered (unranked) tree*, i.e. a parent-child relationship between resources. As an immediate example take the directory structure in a file system. An interesting application of unordered trees is in semistructured data modeling, we will introduce this in detail in Chapter 8. Another interesting hierarchical model may also have order between children of the same parent, in this case we are talking about *ordered trees* whose immediate example is the structure of HTML and XML documents. In some cases XML is used as a formalism to describe semistructured data (e.g. data resulting from integration of different data sources) where the sibling order is not interesting (or it could be even misleading). In addition, document order is not definable is the document is distributed in a P2P (Peer-to-Peer) network. In these cases an unordered model is more likely to be used.

In [42, 37, 32] spatial logics describing unordered labelled trees are presented and studied. These logics are, essentially, static fragments of Ambient Logic [45] and can be used to describe and reason about tree-shaped resources (e.g. semistructured data as we will see in Chapter 8).

Tree-shaped resources are seen as freely generated from (parallel) compositions of trees $F \mid F'$ and locations containing (edge leading to) trees $l[F]$. Mirroring this structure, the logic features, in addition to standard propositional connectives, the parallel (de)composition connective $A \mid B$ and the location connective $l[A]$. The resulting logic is able to describe succinctly structural properties of trees using logical operators lifted from the model constructors. In the last years several uses of spatial tree logic have being investigated including model-checkers, type systems and query languages for tree structured resources.

1.2.1 Unordered Labelled Tree Model

Unordered hierarchical models for spatial logics can be indifferently defined as edge-labelled trees or node-labelled forests. We prefer the edge presentation because it is similar to the one given for ambients, however in [58] an equivalent presentation is given considering forests of node labelled trees.

Finite unordered edge-labelled trees are also called *information trees*, because they represent unordered semistructured information. In this case labels (and their structure) are intended as the *information* provided by the tree.

An information tree (over a label set Λ) is an unordered tree whose edges are labeled over Λ. We define the information tree terms in Table 1.2.1; the description of a tree in this syntax is not unique, for instance the expressions $F \mid F'$ and $F' \mid F$ represent the same unordered tree; similarly, the expressions $\mathbf{0} \mid \mathbf{0}$ and $\mathbf{0}$ represent the same empty tree. We consider two expressions F and F' *congruent* when they represent the same tree, writing $F \equiv F'$. The relation $\equiv$ is an equivalence and a congruence (i.e. a syntactic substitution of equals preserve the equivalence) and it satisfies the axioms reported in Table 1.2.1.

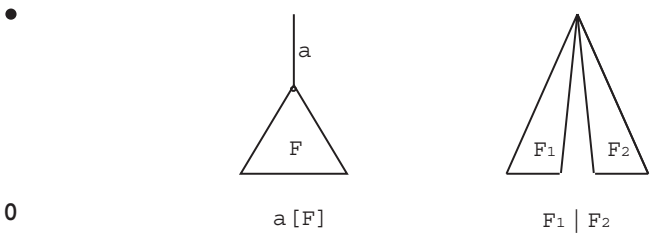

Figure 1.1: Information tree visual representation.

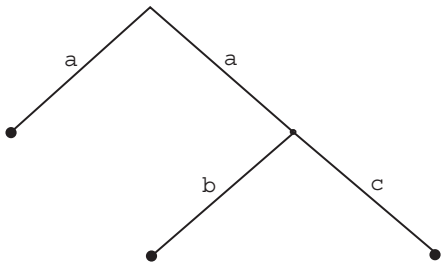

Figure 1.2: An example of information tree.

Table 1.2.1. *Information tree Terms (over Λ) and congruence*

$F, F' ::=$	
$\mathbf{0}$	the empty tree consisting of a single root node
$l[F]$	a single edge tree labeled $l \in \Lambda$ leading to the subtree F
$F \mid F'$	the tree obtained by merging the roots of the trees F and F'
$F \mid \mathbf{0} \equiv F$	neutral element
$F \mid F' \equiv F' \mid F$	commutativity
$(F \mid F') \mid F'' \equiv F \mid (F' \mid F'')$	associativity

In Figure 1.1 we report the standard visual representation of information trees.

Example 1.2.1. *Assume $a, b, c \in \Lambda$, the expression $a[\, b[\mathbf{0}] \mid c[\mathbf{0}]\,] \mid a[\mathbf{0}]$ represents a tree with two edges labeled by a carrying $b[\mathbf{0}] \mid c[\mathbf{0}]$ and the empty tree as children respectively as shown in Figure 1.2.*

In examples and discussions, we will often abbreviate $m[\mathbf{0}]$ as $m[]$, or as m. The structure described here deals with finitely branching information trees only, in [42, 54] infinitely branching information trees are formalized.

A first distinction between the heap model and the tree model is the fact that composition is partial in separation logic models, while the spatial tree logic models

composition is total. In particular it is worth noticing that no constraint or assumption on locations with the same label is performed, i.e. $a[F] \mid a[F]$ is definable and different from both $a[F]$ and $a[F \mid F']$. It may seem a trivial consideration, but actually many models or extensions of ambient logic have some constraints in this case (e.g. the Biri-Galmiche logic, presented in Section 1.3.1, makes the *same-path same-tree* assumption $a[F] \mid a[F'] \equiv a[F \mid F']$ or in some ambient calculi the labels denote sites that are uniquely identified in the same level).

1.2.2 The Logic STL

We define Spatial Tree Logic (STL for short) in Table 1.2.2 as the static propositional logic for unordered trees studied in [32]. The meaning of formulas can be given by a *forcing relation* relating an information tree representation with a formula.

Table 1.2.2. *Propositional Spatial Tree Logic*

$A, B ::=$	formula
$\mathbf{F}$	nothing
$\mathbf{0}$	empty tree
$A \Rightarrow B$	implication
$l[A]$	location
$A@l$	location adjunct
$A \mid B$	composition
$A \triangleright B$	composition adjunct
Forcing relation between information trees and STL formulas	
$F \models \mathbf{F}$	$\stackrel{def}{=}$ never
$F \models \mathbf{0}$	$\stackrel{def}{=}$ $F \equiv \mathbf{0}$
$F \models A \Rightarrow B$	$\stackrel{def}{=}$ $F \models A$ implies $F \models B$
$F \models l[A]$	$\stackrel{def}{=}$ $\exists F'.\, F \equiv l[F']$ and $F' \models A$
$F \models A@l$	$\stackrel{def}{=}$ $l[F] \models A$
$F \models A \mid B$	$\stackrel{def}{=}$ $\exists F_1, F_2.\, F \equiv F_1 \mid F_2$ and $F_1 \models A$ and $F_2 \models B$
$F \models A \triangleright B$	$\stackrel{def}{=}$ $\forall F'.\, F' \models A$ implies $F \mid F' \models B$

Also in this case we can easily derive other classical connectives ($\mathbf{T}$, $A \wedge B$, $A \vee B$, $\neg A$). Notice how the operator $\mid$ and its adjunct $\triangleright$ are similar to the separating conjunction $*$ and separating implication $-\!\!*$ of Separation Logic. Again the world, here an information tree, is split into two separated worlds satisfying the corresponding subformulas.

Example 1.2.2. *Consider a model of information trees representing a bibliography file. We could write a simple formula asserting that "there is at least one* book *edge, leading to at least one* author*, containing exactly one edge labeled* Ghelli*, leading to nothing":*

$$book[author[Ghelli[\mathbf{0}]] \mid \mathbf{T}] \mid \mathbf{T}$$

In general, formulas of this spatial logic can combine connectives talking about the structure with standard propositional connectives. For example the following formula says that "there exists at least a non empty book that does not contain any edge labeled unpublished*".*

$$book[\neg \mathbf{0} \wedge \neg(unpublished[\mathbf{T}] \mid \mathbf{T})] \mid \mathbf{T}$$

The two adjuncts expresses conditional properties on the structure. Location adjunct $A@l$ expresses that when we add an edge labelled l to the current root, the resulting information tree satisfies A. The composition adjunct, called *guarantee*, guarantees that for every possible information tree satisfying A we put in parallel to the current tree, the result will satisfy B.

Example 1.2.3. *As is shown in [32], for each information tree we can easily build a characteristic formula denoting it (up to structural congruence). Assume that $\bar{F}$ is the characteristic formula of F, we have that $F' \models \bar{F} \triangleright (A@l)$ iff $l[F \mid F'] \models A$. That is we can perform a sort of "contextual" reasoning by observing what happens when we put the current model in a particular context (in this case $l[F \mid -]$ with $-$ denoting an hole).*

1.2.3 Somewhere, Recursion and Quantification

The first proposal of a spatial logic for tree shaped resources has been Cardelli and Gordon's Ambient Logic [43]. That logic is more complex than the one we introduced as STL, featuring also quantifiers on names, temporal connectives (that we will discuss after), and a *somewhere modality* $\diamond A$. The extension of STL with quantification over names is in the standard way :

$$F \models \exists x.\, A \stackrel{def}{=} \exists l \in \Lambda.\, F \models A\{x \leftarrow l\}$$

and the universal quantifier may be derived using negation.

With quantification over names types like "a rooted tree leading to something of type A" are easily definable $\exists x.\, x[A]$. More interestingly two labels in different position of the tree may be implicitly constrained to be equal in expressions like $\exists x.\, x[\mathbf{T}] \mid x[\mathbf{T}]$ describing a tree with two first level edges having the same label.

Another kind of quantification is defined in [44] for names protected in the model, we will discuss some properties of spatial logics with standard and hidden quantification over names in Part III.

The somewhere modality is similar to the sometime modality of temporal logics. We assume a subtree/sublocation relation in the obvious way and we say that a tree $F \models \diamond A$ iff there is some subtree of F satisfying A. In [62] it is proved that this connective can be derived if the logic includes a μ recursion on trees. The μ recursion takes inspiration from the μ calculus [83] and is defined as the minimal set of trees satisfying a determined recursively stated property:

$$F \models \mu\xi.A \stackrel{def}{=} F \in fix\lambda\xi.A$$

where fix is the last fixpoint of a function considering sets of information trees ordered by set inclusion.

In [62] is also shown that minimal and maximal fix point coincide when the assertion is restricted to finite models and the recursion is guarded, i.e. all variables appear in a "non-empty" context.

A limited form of the recursion is the (horizontal) Kleene star on trees. $F \models A^*$ iff $F \equiv \mathbf{0}$ or $F \equiv F' \mid F''$ with $F' \models A$ and $F'' \models A^*$. This operator is useful to define regular types on trees and it is easy to derive from the minimal fix-point as follows: $A^* \stackrel{def}{=} \mu\Xi : \mathbf{0} \vee A \mid \Xi$. In [63] the logic STL extended with Kleene Star is studied and related to Presburger Constraints and Presburger automata.

Presburger's constraints

The first order theory of equality on the free group $(\Lambda, +)$ is decidable, and it is also known as Presburger's arithmetic. A possible presentation of Presburger's arithmetic is given by the following grammar, where Exp is an integer expression and ϕ is a Presbuger's formula, also called *Presbuger's constraint*.

Table 1.2.3. *Presburger's Constraints*

$Exp ::=$	expression
a	positive integer constant
N	positive integer variable
$Exp_1 + Exp_2$	addition
$\phi, \psi ::=$	Presburger's Constraint
$(Exp_1 = Exp_2)$	test for equality
$\neg\phi$	negation
$\phi \vee \psi$	disjunction
$\exists N.\phi$	existential quantification

Presburger's constraints allow the definition of flexible, yet decidable, properties over integer, i.e. "the value of X is strictly greater than the value of Y" is expressed using the constraint $\exists Z.X = Y + Z + 1$, and "X is an odd number" using $odd(X) \stackrel{def}{=} \exists Z.X = Z + Z + 1$. In [64] the modal logic TL (Tree Logic) is presented using *multitree automata* (an extension of tree automata that uses Presburger's constraints). Essentially, TL extends the quantifier/recursion-free fragment of our logic to deal with *normalized information trees*. The idea is to rewrite an information tree of the form $n_1[F_1] \mid \ldots \mid n_p[F_p]$ into $1.n_1[F_1] \mid \ldots \mid 1.n_p[F_p]$, and then apply the rewriting rule:

$$a.n[F] \mid b.n[F'] \rightarrow (a+b).n[F] \textbf{ when } F \equiv F'$$

As a result of the rewriting process, we obtain a normalized information tree, that is an information tree with the equivalent subtrees grouped. The spatial operators of our logic are substituted in TL with the following quantification operators based on Presburger's constraints:

$$\begin{array}{ll} \exists N.\phi(N) : N.n[A] & \text{generalized location} \\ \exists N.\phi(N_1, \ldots, N_p) : N_1.A_1 \mid \ldots \mid N_p.A_p & \text{generalized composition} \end{array}$$

where $\phi(N)$ and $\phi(N_1, \ldots, N_p)$ are Presburger's constraints with free variables N and $N_1, \ldots, N_p$ respectively. Using these operators it becomes possible to state spatial properties that make use of Presburger's constraints, such as "there is an odd number of authors": $\exists N.odd(N) : N.author[\mathbf{T}]$.

In [65] another logic dealing with Presburger's constraints is presented, the *Sheaves Logic* (SL). This time the structures described are ordered trees (i.e. XML documents). SL supports both ordered and unordered composition connectives, so it embeds XML Schema (with the & operator) as a subset. The unordered (and so commutative) composition is expressed as in TL with Presburger's constraints; the sequential composition A, B is, essentially, a non-commutative spatial composition operator. Both TL and SL are based on automata for unranked trees with both associative and associative-commutative symbols. These automata can be used to model formulas of SL and TL; in particular in [65, 64] it is shown that both model checking of SL and TL are decidable. In addition, such automata can be extended to any signature involving free function symbols and an arbitrary number of associative and associative-commutative symbols, giving us the promise to model structures for more complicated logics, as well.

In [63] the logic of TL is used to prove a decidability result on STL with Kleene Star. In [98, 108] a Presburger monadic second-order logic (PMSO), an extension of monadic second-order logic (MSO) with Presburger counting constraints, has been proposed to describe semistructured data with counting capabilities, in [15] it is shown that STL with general recursion (STLμ) is more expressive than PMSO and some syntactic restrictions are proposed over STLμ to capture precisely PSMO and even MSO.

1.2.4 STL vs Separation Logic

The similarities between the two logics are obvious. Both have a commutative connective splitting the world into two separated components and both have a corresponding adjunct. But the structural differences in the underlying model is reflected also in the semantics of separating conjunction and parallel composition. Since a composition of two equal models is not allowed in separation logic (two equal addresses are not permitted), than satisfaction of separation logic formulas can be transformed into a first order logic with equalities on addresses [34, 90]. This cannot be done for STL essentially because the trees can be used to *count* and the logic

is more similar to monadic second order logic than to first order one. However, the two logics remain very related since results on one logic have influenced or inspired results on the other. In particular a decidability result was proved for Separation Logic [35] and then extended to Spatial Tree Logic [32]. More recently the results obtained in STL with Kleene Star [63] inspired the other way around, that is some encoding of separation logic [34]. It is clear that a logical framework combining the two approaches in an orthogonal way should be able to better clarify the differences of those models and to unify the similar proofs.

1.3 Separation Logic for Resource Trees

The similarities between spatial and separation logics and the need for new models for structures in memory has led to the investigation of extensions of separation logic to model distributed resource locations and hierarchy. In this section we briefly overview those kind of logics.

1.3.1 Biri-Galmiche Logic

In [12] the bunched implication logic is extended with a modality for locations. The model proposed allows to reason and prove properties of a new data structure, called resource tree, that is a node-labelled tree in which nodes contain resources that belong to a partial monoid (see Section 2.1.1 for a definition of partial monoid).

This resource tree model is different from the one proposed in spatial tree logic (and spatial logic in general). In this case the model is parametric wrt the resource monoid and there is a different notion of parallel composition. In resource tree models parallel composition merges nodes with the same label and composes the other as usual composition of trees. Then, the composition of two nodes with the same label $([l]P \mid [l]Q)$ is equivalent (i.e. the corresponding terms are congruent) to one node which contains resources as subtrees of these two nodes $([l](P \mid Q))$.

We don't want to go into detail on this, we mentioned this logic as an example of another possible approach that BiLog could model. It could be interesting to investigate if BiLog can be instantiated also to model those kind of structures and embed this logic.

1.3.2 A Logic for Trees with dangling pointers

In [39] we have the first proposal that explicitly combines spatial logics and separation logic models. The resulting model is of unordered labelled trees with uniquely identified nodes. Nodes can contain values that refer one other noses using the pointer mechanism (the same seen in the heap). In this model we have two kinds of names:

- the *labels* describing the location as for information trees labels (associated to edges);
- the *identifiers* that identify the resource as heaps addresses (associated to nodes)

In Table 1.3.1 we give the syntax of trees with dangling pointers, the structural congruence $\equiv$ between these terms just specify that the tree branches form a multiset (as for information trees). The main difference wrt information trees is in the partiality of composition, here when two subtrees contain the same label the composition is not defined.

Table 1.3.1. *Trees with dangling pointers*

$T, T' ::=$	trees with pointers
nil	empty tree
$a_n[T]$	a tree labelled a with identifier n and subtree T
$@n$	a pointer to the location identified n
$T * T'$	partial parallel composition

The logic proposed in [39] resembles STL with quantification and recursion. The authors use the logic to express properties of semistructured data, in this sense the logic can be viewed as an extension of the logic of the query language TQL. The logical models studied of [40] and of [38] can be seen as extension of this model with the notion of name abstraction on identifiers.

1.3.3 A Context Logic for Trees

In [33] a spatial context logic is presented to reason about programs manipulating a tree structured memory with node names used to identify memory locations. Terms are unordered labelled trees T and unary contexts of trees C, i.e., trees with one hole.

Table 1.3.2. *Trees with pointers and Tree Contexts*

$T, T' ::=$	trees with pointers
$\mathbf{0}$	empty tree
$a_n[T]$	a tree labelled a with identifier n and subtree T
$T \mid T'$	partial parallel composition
$C ::=$	trees context
$-$	an hole (the identity context)
$a_n[C]$	a tree context labelled a with identifier n and subtree C
$T \mid C$	context right parallel composition
$C \mid T$	context left parallel composition

Correspondingly, the logic is defined by formulas of two kinds: formulas P, which describe trees, and formulas K, which describe tree contexts. Both of these have spatial operators built by using constants (i.e., the empty tree for T and the hole in C), application $K(P)$, and its two adjuncts $K \triangleright P$ and $P_1 \triangleleft P_2$. Formula $K \triangleright P$ represents a tree that satisfies P if inserted in a context satisfying K. Dually, $P_1 \triangleleft P_2$ represents contexts that composed with a tree satisfying P_1 produce a tree satisfying P_2. In Table 1.3.3 we report the logic syntax for the forcing relation definition we refer the reader to [33]. We give only an idea of the semantics of the new operators. $K(P)$ is satisfied by a tree T when there exist a tree context C and a subtree T' such that $T \equiv C(T)$, C satisfies the context formula K, and T' satisfies the tree formula P. Notice that the context application formula is not commutative (as the binary operators for spatial logics we have seen till now), thus it admits two different adjuncts. These adjuncts correspond to $\triangleleft$ (quantifying on contexts) and $\triangleright$ (quantifying on trees). It will be proved in Section 4.4.1 that this logic can be naturally embedded in an instance of BiLog.

Table 1.3.3. *Context Tree Logic (CTL)*

$P, P' ::=$	tree formulas
$false$	
$K(P)$	context application
$K \triangleleft P$	context application adjunct
$P \Rightarrow P'$	implication
$C, C' ::=$	context formulas
$false$	
$-$	identity context formula
$a_x[-]$	node context formula
$P \triangleright P'$	context application adjunct
$P \mid -$	parallel context formula
$P \Rightarrow P'$	implication

1.4 Spatial Logic for Graphs

Another proposal for describing semistructured resources was presented in [40] a spatial logic to describe properties of labeled directed graphs is presented. The structure modeled is a directed graph with labeled edges and named nodes. The graph logic combines standard first-order logic with additional structural connectives: the composition of graphs $G \mid G'$, and the basic edge $a(n, m)$ where a is the edge label and n, m are node names. With these connectives, and quantification over names, we are able to express properties such as "there are three connected edges labeled a, b, a":

$$\exists x, y, z, u.\ a(x, y) \mid b(y, z) \mid a(z, u)$$

or "there is a path $a.b.a$":

$$\exists x, y, z, u.\ (a(x, y) \mid \mathbf{T}) \wedge (b(y, z) \mid \mathbf{T}) \wedge (a(z, u) \mid \mathbf{T})$$

The logics of [40] also includes edge label quantifier and recursion. In [40] this logic is used as a pattern matching mechanism of a query language for graphs. In addition the logic is integrated with *transducers* to allow graph transformations. There are many applications of this graph logic, including semistructured data description and manipulation. Additional studies on expressivity and complexity of this logic are in [67].

1.5 Describing Processes

Althought this Thesis is mostly devoted to static models, we will also investigate some aspect of dynamics. Actually dynamics is the most promising direction of BiLog, the spatial logic framework we will propose in Part II, thanks to the theory of bigraphs, the model BiLog was inspired by. In this section we give an introduction on process algebras and we briefly overview the spatial logic proposals in this context.

1.5.1 Basics on Process Calculi

In the first half of the 20th century, various formalisms were proposed to capture the informal concept of computable function, μ-recursive functions, Turing Machines and the λ-calculus possibly being the most well-known examples today. The surprising fact that they are essentially equivalent is the content of the Church-Turing thesis. Another shared feature is more rarely commented on: they all are most readily understood as models of sequential computation.

The subsequent consolidation of computer science required a more subtle formulation of the notion of computation, in particular explicit representations of concurrency and communication. Petri-Nets and calculi such as Tony Hoare's CSP, Robin Milner's CCS and the π-calculus by Milner, Joachim Parrow and David Walker are currently the most prominent calculi to have emerged from this line of research.

The process calculus approach gathered momentum in the 1970s when it became increasingly clear that the then dominant approaches to modeling computation were unlikely to yield satisfactory accounts of non-deterministic, non-terminating and interacting agents. Instead, early pioneers such as Milner and Hoare decided to make interaction between agents executing in parallel the basic computational primitive. This can be done in several ways that can be characterised by three core design decisions.

- Computing agents have zero or more discrete points of connection and interaction called, interchangeably, names, channels or interaction points. Parallel composition of two agents involves connecting their interaction points by links,

whenever they share a name. Crucially, a link does not preclude further connections at a name. The Internet is a good source of analogies here: names correspond roughly to IP addresses (plus port numbers, but let's ignore this detail for simplicity).

- Computation itself is binary, point-to-point interaction between independent agents. Interaction happens along names by handshaking or synchronisation between a sender and a receiver. This handshaking may or may not involve passing data from the sender to the receiver. In the Internet, interaction happens by sending IP packets between computers.

- Interaction is an atemporal event in the sense that it does not have a duration. Interactions may be ordered in time. Here the Internet analogy breaks down because the duration of packet delivery from sender to receiver has important semantic consequences.

To define a process calculus, one starts with a set of names, the discrete interaction points. Names have no internal structure apart from what is required to distinguish names from one another. Hence names are pure. Their only purpose is to denote interaction points. In many implementations, names have rich internal structure to improve efficiency, but this is abstracted away in most theoretic models. In addition to names, one needs a means to form new processes from old: the crucial operators, always present in some form or other, allow the parallel composition of processes, to specify which channels to use for sending and receiving data, sequentialisation of interactions, hiding of interaction points and recursion or replication. Parallel composition of two processes P and Q, usually written $P \mid Q$ is the key primitive of process calculi. It allows computation in P and Q to proceed simultaneously and independently. But it also allows interaction, that is synchronisation and flow of information from P to Q on a channel shared by both (or vice versa). Channels are 'created' by shared names in parallel composition.

Interaction is a directed flow of information. That means, input and output are distinguished as dual interaction primitives. We have an input operator $x(v)$ and an output operator $x\langle y\rangle$, both of which name an interaction point (here x) that is used to synchronise with a dual interaction primitive. Should information be exchanged, it will flow from the outputting to the inputting process. The output primitive will specify the data to be sent. In $x\langle y\rangle$, this data is y. Similarly, if an input expects to receive data, one or more bound variables will act as place-holders to be substituted by data, when it arrives. In $x(v)$, v plays that role. But what kind of data is exchanged in an interaction? There are various choices. It will turn out that this choice is a key distinguishing feature between process calculi.

Sometimes interactions must be temporally ordered, because we might want to specify algorithms like: first receive some data on x and then send that data on y. Sequential composition can be used for such purposes. It is well-known from other models of computation. In process calculi, the sequentialisation operator is

usually integrated with input or output or both. For example the process $x(v).P$ will wait for an input on x. Only when this input occurs, P will be activated with the received data substituted for v.

The key operational rule, containing the computational essence of process calculi, can be given solely in terms of parallel composition, sequentialisation, input and output: although the details vary, it always looks something like this.

$$x\,\langle y\rangle\,.P \mid x(v).Q \longrightarrow P \mid Q\{v \leftarrow y\}$$

The process $x\,\langle y\rangle\,.P$ sends a message, here y, along the channel x. Once that message has been sent, $x < y > .P$ becomes the process P. Dually, the process $x(v).Q$ receives that message on channel x to become $Q\{v \leftarrow y\}$, which is Q with the placeholder v substituted by y, the data received on x. The class of processes that P is allowed to range over as the continuation of the output operation substantially influences the properties of the calculus.

Processes do not limit the number of connection that can be made at a given interaction point. But interaction points allow interference (i.e. interaction). For the synthesis of compact, minimal and compositional systems, the ability to restrict interference is crucial. Hiding operations allow to control the connections made between interaction points when composing agents in parallel. In sequential models of computation, scoping rules, procedures and objects facilitate hiding (but they might have other uses, too: functional features in the case of procedures and subtyping with its associated dispatch mechanisms for objects). We denote the hiding of a name x in P by $(\nu x)P$.

Many different variants of process calculi have been studied and not all of them fit the paradigm sketched here. The most prominent example may be the Ambient calculus [47]. Ambient Calculus includes the notion of *ambient* $a[P]$ as a spatial location named a that can move from one hosting location to another. The location structure is tree-shaped (like information trees) and processes may also have forms specifying the behaviour (like $in\,a.P$ or $out\,a.P$). In ambients calculi we have no interaction points for communications, thus input and output processes are of the form $(x).P$ and $\langle y\rangle\,.P$ and the communication in is allowed if input and output operations are located inside the same ambient.

Another example of an ambient computation is the mobility, e.g.

$$in\,a.Q \mid a[P] \longrightarrow a[P \mid Q]$$

1.5.2 Extensional Logics

The knowledge we have on objects or systems is commonly said *intensional* if we are able to 'look inside' them and discover their internal mechanism, from this we may be able to understand or derive every possible future behaviour of the system. On the other hand, if we cannot 'look inside' the object, the only thing we are able to do is to 'observe' how the object interacts with the environment with several

experiences. The kind of knowledge obtained in this way is said *extensional*. If we apply this concepts to process calculi, the intensional description corresponds to know exactly the structure of the process (the way it is implemented), while the extensional description corresponds to know the observable behaviour in the possible experiences. The classic approach is to consider as possible observable experience every composition with any process. If we are intensional, two process description are equivalent iff they are structurally the same (i.e. structural congruence is the intensional equivalence), while in the extensional case two process description are equivalent if they behave the same. We say that a logic is extensional if its logical equivalence corresponds to an extensional equivalence (it can be bisimulation or a sort of *barbed* congruence). In other cases, in particular when logical equivalence is structural congruence, the logic is said to be intensional.

The main example of extensional logic for processes is the Hennessy-Milner logic []. It is an extension of the classic logic with a modality $< \alpha > A$ for each possible label α. The semantics is given wrt a labelled transition system

$$P \models < \alpha > A \iff \exists P'. P \xrightarrow{\alpha} P' \wedge P' \models A$$

Recently an extensional fragment of Ambient Logic has been studied in [74].

1.5.3 Intensional Logics

Logics for concurrent systems are certainly not new, but the intent to describe spatial properties seems to have arisen only recently. The first spatial logic for concurrency is the Ambient Logic, introduced by Cardelli and Gordon in [43] to reason about mobility in distributed and concurrent system with named locations. The structures described in this logic are processes, replicable trees that can evolve over time. In [107] the logical equivalence induced by the Ambient Logic is studied and it is proved that Ambient Logic is intensional, in the sense that it can observe structural properties of the models that are not observable in behaviour.

In [44] the Ambient Logic is extended with the *name restriction* of processes $(\nu n)P$. The notion of name restriction was introduced in π-calculus to represent hidden communication channels. In the context of the ambient calculus, the name restriction can be used to represent hidden locations and secret locations. In [44] properties on structures with name restriction are expressed introducing in the logic the *revelation* and *hiding* connectives; the notion of fresh-name quantifier is also introduced.

In [28] the study on restriction is advanced in a spatial logic for processes with no locations. This logic includes recursion, second order quantification and fresh-name quantification; recursion is, indeed, derived throught second order quantification. In [29] a new presentation style for the rules of spatial logics is introduced: the logic of [28] is presented as a modal sequent calculus with "world constraints".

Here we present a minimal fragment only with the aim to give an idea of a dynamic spatial logic, we will show how to embed this fragment in BiLog in Section4.5.

A Minimal Dynamic Spatial Logic

The work [31] introduces the spatial logic $\mathcal{L}_{spat}$ suitable to describe the structure and the behaviour of CCS processes. The language of the logic is

$$A, B \quad ::= \quad 0 \quad | \quad A \wedge B \quad | \quad A \mid B \quad | \quad \neg A \quad | \quad A \triangleright B \quad | \quad \Diamond A.$$

It includes the basic spatial operators: the void constant 0, the composition operator $|$, and its adjunct operator $\triangleright$. It presents also a temporal operator, the next step modality $\Diamond$, to capture the dynamics of the processes. The paper [31] defines a semantics to $\mathcal{L}_{spat}$ in term of CCS processes, as outlined in Table 1.5.1. In particular, the parallel connective describes processes that are produced by the parallel between two processes that satisfy the corresponding formula. A process satisfies the formula $A \triangleright B$ if it satisfied the formula B whenever put in parallel with a process satisfying A. Finally the next step $\Diamond A$ is satisfied by a process that can evolve into a process satisfying A.

Table 1.5.1. *Semantics of formulas $\mathcal{L}_{spat}$ in CCS*

$P \models_{spat}$	0	if $P \equiv \mathbf{0}$
$P \models_{spat}$	$\neg A$	if not $P \models_{spat} A$
$P \models_{spat}$	$A \wedge B$	if $P \models_{spat} A$ and $P \models_{spat} B$
$P \models_{spat}$	$A \mid B$	if there exist R and Q, s.t. $P \equiv R \mid Q$, $R \models_{spat} A$ and $Q \models_{spat} B$
$P \models_{spat}$	$A \triangleright B$	if for every Q, $Q \models_{spat} A$ implies $P \mid Q \models_{spat} B$
$P \models_{spat}$	$\Diamond A$	if there exist P' s.t. $P \longrightarrow P'$ and $P' \models_{spat} A$

1.6 Decision Problems in Spatial Logics

In this section we describe the decision problems that arise in Spatial Logics and we give an overview on the main known results about their decidability. Another open question is the expressivity and minimality of spatial logics, these have been deeply investigated in [89].

Model checking

The forcing relation of a spatial logic induces immediately the following decision problem: Given a model P and a formula A is it decidable to check whether $P \models A$?

A decision procedure for such problem is also called a *model checking* algorithm. As an example a model checking algorithm for STL implements a matching procedure between an information tree and a STL formula, where the result of the match is just success or failure. For example, the following match succeeds:

$$book[year[1999] \mid author[Ghelli] \mid \ldots] \mid book[\ldots] \models book[author[Ghelli[\mathbf{0}]] \mid \mathbf{T}] \mid \mathbf{T}$$

In this case the matching process simply must verify if there is a *book* containing an *author* edge leading to the singleton *Ghelli*. More generally, considering free variables in the formula as *matching variables*, we can collect information during the matching process and bind it to the variables; the result of the matching algorithm is either the failure or an association of matching variables to the trees that match them. That is the basic idea for the binding mechanism of the TQL query language; we will introduce TQL (and its binding mechanism) in Section 8.2.2.

Validity problem

Spatial logics describe properties of structures (e.g. processes or trees). With the satisfaction relation we match a particular structure against a formula (a property description). A more general problem is whether a formula is *valid*, i.e. it is matched by every possible structure. A closely related question is the *satisfiability*: "is there any structure matching the formula?". Most problems to reason about structures can be rephrased as validity or satisfiability problems, so algorithms deciding validity are very useful.

Some Known Decidability Results

In [32] it is proved that validity and model-checking problems are, in fact, equivalent in STL and it is shown how to decide them by model checking and, alternatively, by deduction in a sequent calculus. Notice that neither model checking nor validity is obviously decidable when adjunct operators like $A \triangleright B$ are present in the logic, since their forcing relation is defined as an infinite quantification over all the terms. Validity, satisfaction and satisfiability for a decidable sublogic of the Ambient Logic are investigated in [43], where a model checking algorithm is given for the logic without guarantee ($\triangleright$). In [63] it is proved that STL plus the star recursion operator is decidable. In [41] a decision procedure is presented for query answering in STL without adjuncts and with quantification and tree variables. The model cheking in TQL and its complexity is studied also in [13]. In [50] it is proved that if we add existential quantification to a simple spatial logic the validity becomes undecidable. More recent results are presented in [15] and [14] where the static spatial tree logic model cheking with μ-reursion is studied in detail and compared to the MSO a Prebrurger MSO of [109], [110], and [63].

Chapter 2

Bigraphs

In this chapter we will introduce briefly the theory and axiomatization of bigraphs, a new meta-model and calculus for concurrent distributed and mobile systems recently proposed by Robin Milner et al. [79].

There are two main challenging motivations inspiring the research on bigraph theory:

Global Computing The long term challenge of bigraph theory is to model computation on a global scale, like Internet and World Wide Web. The aim is not only to describe existing systems, but also to specify and design new systems from sketch and manage running systems adaptations. There are many aspects of global computing to be modeled and the scientific approach commonly applied is to develop separate theories for each specific aspect. Bigraph's aim is to tackle two aspects of mobile systems simultaneously and orthogonally: *mobile locality* and *mobile connectivity*. Another important aspect to consider is the *open-endedness*, i.e. the system described can interact with external parts (that can be both located "outside" or "inside"). A formal treatment of this three aspects seems to require new mathematical structures, and bigraphs attempt to provide them in an extensible way.

Theory Unification The other main challenge is more theoretical and foundational: "to provide a theory common to different process calculi, and to base this theory on the topographical ideas that appear to pervade these calculi" (Robin Milner). In particular the main point is to find a uniform theory for the behaviour, so that many process calculi can be expressed in the same frame without seriously affecting their treatment of behaviour. The work in this direction was started with action calculi [92] that was afterward simplified in order to treat *locality* and *connectivity* independently.

Bigraphs use many ideas from many sources: the Chemical Abstract machine (Cham) of Berry and Boudol [11], the π-calculus of Milner, Parrow and Walker [97], the interaction nets of Lafont [86], the mobile ambients of Cardelli and Gordon [47],

the explicit fusions of Gardner and Wischik [72] developed from the fusion calculus of Parrow and Victor [112], Nomadic Pict by Wojciechowski and Sewell [113], and the uniform approach to a behavioural theory of reactive systems of Leifer and Milner [88].

In this chapter we introduce in some detail the pure bigraphical structure (both concrete and abstract). We also briefly present the some bigraph refinements. Most concepts presented in this chapter will be used in Part II to describe models of our logical framework.

2.1 Pure Bigraphs

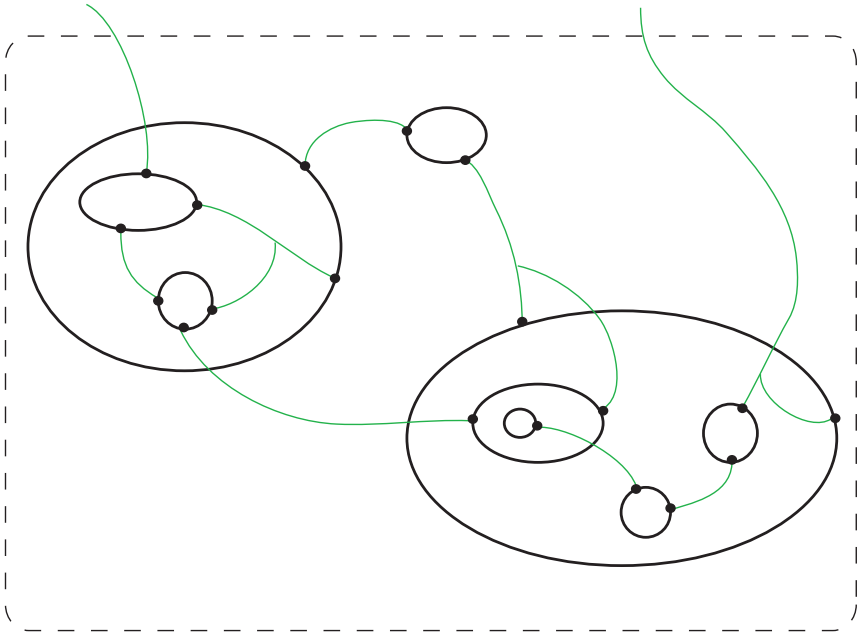

Figure 2.1: A bigraph: nested and connected nodes

The diagram in Figure 2.1 shows a pure bigraph, avoiding some of its details. The ovals and circles are *nodes* which may be nested, and each node has *ports* (shown as dots on the node perimeter) which may be linked. A bigraph represents an open-ended system, this can be seen already in Figure 2.1 where the wires escaping from the top of the diagram represent external links. When this bigraph is inserted in another (insertion will be represented by categorical composition) it will be placed in some *region* of that host graph, and each external link joined to some link of the host in a way that does not depend on the placing.

Pure bigraphs are the core of bigraph theory. They formalise distributed systems maintaining orthogonality between their main characteristics: locality and intercon-

nections. The set of *nodes* in a pure bigraph is shared between two structures: the so-called *place graph* modeling the hierarchical nesting in a tree structure, and the so-called *link graph* modeling the connections of the node ports to each other (and to names) in a hyper-graph structure. Place graphs express locality, i.e., the physical arrangement of the nodes. Link graphs are hyper-graphs and formalise connections among nodes. The orthogonality of the two structures dictates that nesting imposes no constraint upon interconnections. Constraints may be added in refinements of pure bigraph theory, as *binding* bigraphs that will be introduced in section 2.3.

2.1.1 Preliminars

Graphically bigraphs are simply pairs of graphs on the same nodes, we need however to study them in a mathematical setting in order to formally describe their structure and to deal with dynamics in an uniform way. We are going to describe bigraphs and their constituent substructures as arrows in (some kinds of) *monoidal categories*. For the sake of self-containment we give now a very brief introduction on categories. It aims to present only the basic concepts and notations needed to have an idea of the theory underlying bigraphs.

Notations

We denote natural numbers $m, n, \ldots$ and we interpret them as finite ordinals, e.g. $m = \{0, 1, \ldots, m-1\}$. We presuppose a *denumerable* set of names Λ and we will denote finite sets of names X, Y with $X, Y \subset \Lambda$. We denote $\vec{a}$ a finite sequence $\{a_i \mid i \in m\}$. We denote $\uplus$ the disjoint union.

On Categories

In mathematics, categories are used to formalize notions involving abstract structure and processes which preserve structure. Categories appear in virtually every branch of modern mathematics and are a central unifying notion. The study of categories in their own right is known as *category theory*. For an extensive treatment of category theory we redirect the reader to [91].

Essentially, a category C consists of a class $ob(C)$ of objects and a class $hom(C)$ of morphisms. Each morphism f has a unique source object a and target object b. We write $f : a \to b$, and we say f is a morphism from a to b. We write $hom(a, b)$ to denote the hom-class of all morphisms from a to b. For every three objects a, b and c, the binary operation $hom(a, b) \times hom(b, c) \to hom(a, c)$ is called composition of morphisms; the composition of $f : a \to b$ and $g : b \to c$ is written as $g \circ f$ or gf, such that the following axioms hold:

(associativity) if $f : a \to b$, $g : b \to c$ and $h : c \to d$ then $h \circ (g \circ f) = (h \circ g) \circ f$,

(identity) for every object x, there exists a morphism $id_x : x \to x$ called the identity morphism for x, such that for every morphism $f : a \to b$, we have $id_b \circ f = f = f \circ id_a$.

From these axioms, one can prove that there is exactly one identity morphism for every object.

We will refer to morphisms of a category as *arrows* due to their visual representation in commutative diagrams.

Example 2.1.1. *We present a category in terms of its objects, its arrows (morphisms) and its composition of arrows. The classical example is the category* **Set** *having sets as objects, functions as arrows and the composition of arrows is the classical function composition. Many mathematical concepts such as* groups, vector spaces *or* topological spaces *may be seen as subcategories of the category* **Set**, *i.e. they are categories obtained adding more structure to sets and by requiring that morphisms (in this case functions) respect this structure.*

A *monoid* is a tuple $(M, \otimes, \epsilon)$ where M is a set, $\otimes$ is an associative binary operation $\otimes$: $M \times M \to M$ (called *tensor product*), and ϵ is a particular element of M behaving as an identity wrt $\otimes$, i.e. $x \otimes \epsilon = x = \epsilon \otimes x$ for each $x \in M$. If the binary operation is partial (but remains associative when defined), we can call the structure a *partial monoid*. If the binary operation is commutative (i.e. $x \otimes y = y \otimes x$) the structure is called a *commutative monoid*.

A monoidal category is a category having a monoid structure $(M, \otimes, \epsilon)$ on the objects that is respected by the morphisms. This means that there will be also an $\otimes$ operator on the arrows such that:

$f \otimes id_\epsilon = f = id_\epsilon \otimes f$	identity
$f \otimes (g \otimes h) = (f \otimes g) \otimes h$	associativity
$id_a \otimes id_b = id_{a \otimes b}$	identity tensor
$(f \otimes g) \circ (f' \otimes g') = (f \circ f') \otimes (g \circ g')$	bifunctoriality

We refer to the last axiom as *bifunctoriality property*; we will study in Part II logical models observing these axioms and we will refer to this kinds of models (and their elements) as bifunctorial.

When the monoid is partial a (not really common) notion of *partial monoidal category* arise. In this case the axioms are required to hold only when both sides are defined.

A monoidal category is called *symmetric* if its tensor product is not only associative but also commutative up to suitable natural isomorphisms $\gamma_{a,b} : a \otimes b \to b \otimes a$. γ must satisfy:

$$\gamma_{a,\epsilon} = id_a$$
$$\gamma_{a,b} \circ \gamma_{b,a} = id_{a \otimes b}$$
$$\gamma_{a,b} \circ (f \otimes g) = (g \otimes f) \circ \gamma_{a',b'}$$

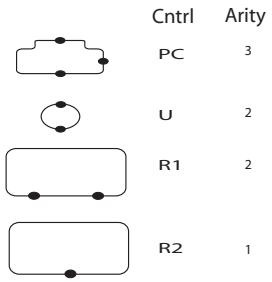

Figure 2.2: An example signature.

2.1.2 Definitions

First of all we define the bigraphical signature describing the possible controls with associated arities representing the number of ports. Controls are naturally associated to the *shape* of nodes in the graphical visualization of bigraphs.

Definition 2.1.2 (Signature)**.** *A signature $\mathcal{K}$ is a set whose element are called controls. For each control $K \in \mathcal{K}$ it provides a finite ordinal $ar(K)$, an arity.*

Notice that, since arity is an ordinal, the ports are ordered and it makes sense to identify a port as the n-th port of the corresponding control.

Example 2.1.3. *The signature may be chosen according to the context we want to describe. As a possible example we can imagine to model locations and connections in a system with people and computers interacting. We fix the signature to be $\mathcal{K} = \{\mathsf{PC}, \mathsf{U}, \mathsf{R1}, \mathsf{R2}\}$, where PC represents a computer, U an user, and $\mathsf{R1}$ and $\mathsf{R2}$ two different kinds of office rooms. To complete the signature we have to specify the arities of controls. In this case we fix $ar(\mathsf{PC}) = 3$, $ar(\mathsf{U}) = 2$, $ar(\mathsf{R1}) = 2$, and $ar(\mathsf{R2}) = 1$. In Figure 2.2 we give a graphical idea of this signature.*

In refinement of the theory a signature may carry further information, such as a *sign* or a *type* for each port. Another possible refinement is to add *kinds* to nodes, determining the controls a node may contain or specifying how the control influences the behaviour of its content.

Example 2.1.4. *Consider the signature of Example 2.1.3, we may want to differentiate physical and virtual ports. The three computers ports can represent two physical connection ports to the power and LAN respectively and a virtual connection. The users have two virtual connection ports and rooms $\mathsf{R1}$ and $\mathsf{R2}$ provide physical ports (e.g. power plugs). To constraint physical ports to be connected to physical ports only (and the same for virtual ports) we may assign types 'p' and 'v' to ports and specify a sorting discipline accordingly. In addition we may want controls U and PC to have an **atomic** kind, i.e. to not contain any other control, while controls $\mathsf{R1}$ and $\mathsf{R2}$ may have a **not-atomic** kind. Alternatively we can constrain rooms to contain*

at least a PC *assigning them a particular kind and specifying a sorting discipline accordingly.*

We will introduce sorting disciplines and refinements of bigraphs in section 2.2.

Now we are ready to define the concrete structures (presented in [80]) constituent of bigraphs. These concrete instances identify nodes in order to describe the relations between them. This identification is not captured by the abstract graphical concept (nodes are not necessarily identified in diagrams) and the abstract notion can be obtained as a lifting of this structure that *forgets* the identifiers. The identification of nodes is necessary to relate the two substructures (link and place graphs) and, more importantly, to guarantee the existence of Relative Push Outs in the dynamic theory.

Definition 2.1.5 (Concrete Place Graph)**.** *$P = (V, ctrl, prnt) : m \to n$ is a concrete place graph over the signature $\mathcal{K}$ having 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 each $k > 0, v \in V$.*

A place graph is actually a ordered forest of nodes with associated controls (due to the aciclicity constraint). A place graph $P : m \to n$ features a number of roots equal to n and has m 'holes' appearing as leafs in the forest. Composition $P \circ P'$ corresponds to place the roots of P'(in order) in the corresponding holes of P. It is defined only when nodes are disjoint. The product $P \otimes P'$ is simply the place graph obtained putting P an P' one next to the other, also in this case it is defined when node sets are disjoint only.

Definition 2.1.6 (Concrete Link Graph)**.** *$W = (V, E, ctrl, link) : X \to Y$ is a concrete link graph over the signature $\mathcal{K}$ having a 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 $link : X \uplus Ports \to E \uplus Y$ called the* link map*, where the disjoint sum $Ports = \sum_{v \in V} ar(ctrl(v))$ is the set of* ports *of W.*

A *point* of a link graph is either a port or a inner name. A *link* of a link graph is either an edge or an outer name. Essentially, a link graph associates points to links. Composition of link graphs $W \circ W'$ corresponds to 'connect' the inner names of W with outer names of W' resulting in a link graph having as nodes the union of the set of nodes, as edges the union of the set of edges. Also in this case the product is simply putting two link graphs one next to the other.

Notice that edges are not constrained to really connect something. The edges not connected are called *idle edges*. Idle edges are modeled in link graphs because, in presence of dynamics, it is possible that nodes disconnect leaving dangling edges.

Definition 2.1.7 (Concrete Pure Bigraph)**.** *$G = (V, E, ctrl, P, W) : I \to J$ is a concrete pure bigraph over the signature $\mathcal{K}$ having $I = \langle m, X \rangle$ and $J = \langle n, Y \rangle$ as*

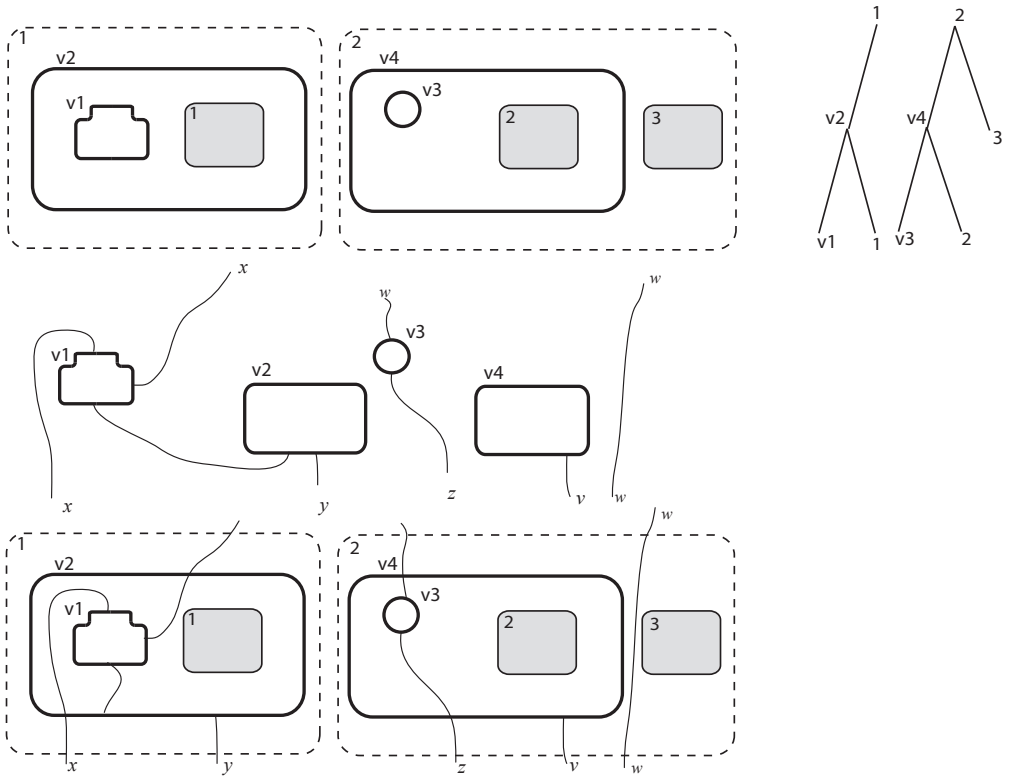

Figure 2.3: A concrete pure bigraphs and its link and place graphs

its inner *and* outer faces. *V and E are finite sets of* nodes *and* edges *respectively and together with the* control map *$ctrl : V \to \mathcal{K}$ are shared by the place graph $P = (V, ctrl, prnt) : m \to n$ and the link graph $W = (V, E, ctrl, link) : X \to Y$.*

In Figure 2.3 we show a concrete bigraph with its corresponding concrete place and link graphs.

An *abstract bigraph* is an equivalence class of concrete bigraphs considering only the structure and ignoring node names, i.e. two concrete bigraphs represent the same abstract bigraph if they differ only in a bijection on their nodes and non-idle edges; idle edges are ignored.

We introduced concrete bigraphs to give an idea of how to implement bigraphs, however for the remainder of this Thesis we are concerned (when not explicitly contrarily stated) only with abstract bigraphs, that is the equivalent classes of concrete ones. Thus, for now on we will depict bigraphs without node and edge names.

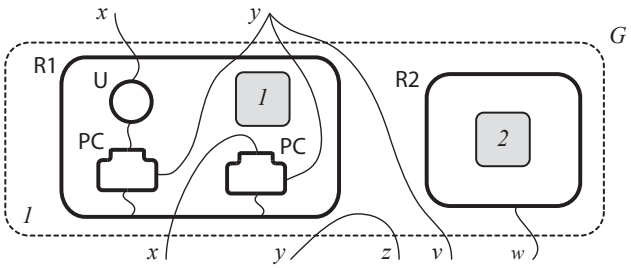

Figure 2.4: A bigraph $G : \langle 2, \{x, y, z, v, w\}\rangle \to \langle 1, \{x, y\}\rangle$.

2.2 Abstract bigraphs example

The bigraph G of Fig. 2.4 represents a system where people and things interact. We imagine two offices with employees logged on PCs. Every entity is represented by a node, shown with bold outlines, and every node is associated with a *control* (either PC, U, R1, R2). Controls represent kinds of nodes, and have fixed *arities* that determine their number of ports. Control PC marks nodes representing computers, and its arity is 3: in clockwise order, these ports represent a keyboard interacting with an employee U, a LAN to an other PC and open to the outside network, and a plug connecting the computer to the electrical mains of office R. Employees U may communicate with each other via the upper port in the picture. The nesting of nodes (place graph) is shown by the inclusion of nodes into each other; the connections (link graph) are drawn like lines.

At the top level of the nesting structure sit the *regions*. In Fig. 2.4 there is one sole region (the dotted box). Inside nodes there may be 'context' *holes*, drawn as shaded boxes, which are uniquely identified by ordinals. In figure the hole marked by 1 represents the possibility for another user U to get into office R1 and sit in front of a PC. The hole marked by 2 represents the possibility to plug a subsystem inside office R2.

Place graphs as arrows

Place graphs can be seen as *arrows* over a symmetric monoidal category whose objects are finite ordinals. We write $P : m \to n$ to indicate a place graph P with m holes and n regions. In Fig. 2.4, the place graph of G is of type $2 \to 1$. Given place graphs P_1, P_2, their composition $P_1 \circ P_2$ is defined only if the holes of P_1 are as many as the regions of P_2, and amounts to *filling* holes with regions, according to the number each carries. The tensor product $P_1 \otimes P_2$ is not commutative, as it 'renumbers' regions and holes 'from left to right'.

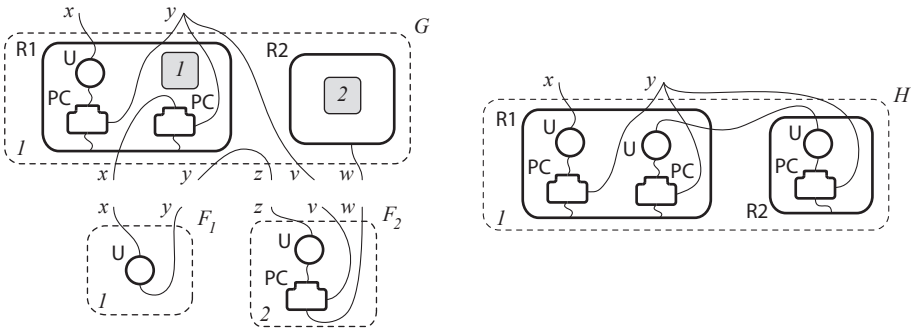

Figure 2.5: Bigraphical composition, $H \equiv G \circ (F_1 \otimes F_2)$.

Link graphs as arrows

Given a *denumerable* set of names Λ, link graphs can be seen as arrows of a partial monoidal category whose objects are (finite) sets of names, i.e. a link graph is an arrow $X \to Y$, with $X, Y \subseteq \Lambda$. The set X represents the *inner* names (drawn at the bottom of the bigraph) and Y represents the set of *outer* names (drawn on the top). The link graph connects ports to names and to other ports, in any finite number. A link to a name is *open*, i.e., it may be connected to other nodes as an effect of composition. A link to an edge (represented in Fig. 2.4 by a line between nodes) is *closed*, as it cannot be further connected to ports. Thus, edges are *private*, or hidden, connections. The composition of link graphs $W \circ W'$ corresponds to *linking* the inner names of W with the corresponding outer names of W' and forgetting about their identities. As a consequence, the outer names of W' (resp. inner names of W) are not necessarily inner (resp. outer) names of $W \circ W'$, and the link graphs can perform substitution and renaming. The tensor product of link graphs is defined in the obvious way only if their inner (resp. outer) names are disjoint.

Pure Bigraphs as arrows

Combining ordinals with names we obtain *bigraphical interfaces*, i.e., pairs $\langle m, X \rangle$ where m is an ordinal and X is a set of names. Combining the notion of place graph and link graphs on the same nodes we obtain the notion of bigraphs, i.e., arrows $G : \langle m, X \rangle \to \langle n, Y \rangle$.

Fig. 2.5 represents a more complex situation. At the top left-hand side is the system of Fig. 2.4. At the bottom left-hand side F_1 represents a user U ready to interact with a PC or with some other users, F_2 represents a user logged on its laptop, ready to communicate with other users. The system with F_1 and F_2 represents the tensor product $F = F_1 \otimes F_2$. The right-hand side of Fig. 2.5 represents the composition $G \circ F$. The idea is to insert F into the context G. The operation is partially defined, since it requires the inner names and the number of holes of G to match the outer names and the number of regions of F, respectively. Shared names

create the new links between the two structures. Intuitively, composition *first* places every region of F in the proper hole of G (place composition) and *then* joins equal inner names of G and outer names of F (link composition). In the example, as a consequence of the composition the user in the first region of F is logged on PC, the user in the second region of F is in room $\mathsf{R2}$. Moreover note the edge connecting the inner names y,z in G, its presence produces a link between the two internal nodes U of F after the composition. We imagine a phone call between the two users.

2.3 Bigraph refinements

Pure bigraphs are the core structure the emerging field of bigraph theory. Many refinements of bigraphs have been (or are being) proposed to model specific structures like π-calculus processes or Petri-Nets.

2.3.1 Binding bigraphs

Pure bigraphs leave connections and locations completely orthogonal. However, in many cases, a locality constraint on the connections is desirable. That corresponds graphically to give 'ports' also to holes. Actually this means that inner and outer faces elements can be located to. Binding bigraphs can be derived from pure bigraphs with a sorting discipline, but they remain interesting. In [80] binding bigraphs are formally defined and used to encode π-calculus processes.

An axiomatization on binding bigraphs has been recently proposed in [66].

2.3.2 Sorted bigraphs

In the following Θ will denote a non-empty set of sorts, and θ will range over Θ.

Definition 2.3.1. *A signature $\mathcal{K}$ is Θ-sorted if it is enriched by an assignment of a sort $\theta \in \Theta$ to each $i \in ar(\mathsf{K})$ for each control K. An interface X is Θ-sorted if it is enriched by ascribing a sort to each name $x \in X$. A bigraph is Θ-sorted over $\mathcal{K}$ if its interfaces are Θ-sorted, and for each K, i the sort assigned by $\mathcal{K}$ to $i \in ar(\mathsf{K})$ is ascribed to the i^{th} port of every K-node.*

We say sorted instead of Θ-sorted when Θ is understood. We may wish to consider only those sorted link graphs that obey some condition:

Definition 2.3.2. *A sorting (discipline) is a triple $\Sigma = (\Theta, \mathcal{K}, \Phi)$ where $\mathcal{K}$ is Θ-sorted, and Φ is a condition on Θ-sorted link graphs over $\mathcal{K}$. The condition Φ must be satisfied by the identities and preserved by both composition and tensor product.*

We shall often say well-sorted instead of Σ-sorted when Σ is understood. Even with only a single sort there are important examples; one example is undirected

linear link graphs, where every open link contains exactly one point, and every closed link exactly two points.

In [95] a sorted link graphs have been used to encode Petri-Nets. In [99] sorted bigraphs static theory is further investigated.

2.4 Term Algebra

The definition of abstract bigraphs in terms of equivalence classes of concrete bigraph is not convenient for practical uses. One would prefer to identify an abstract bigraph as a term of a particular algebra with given operators. These terms could be used in a programming/query language (as in [73]) or as elements of model for a spatial logic (as we will do in Part II). An important tool for understanding and proving properties in a term algebra is finding a *normal form* for terms, that is a unique representation for terms that represent the same structure.

In [94], two different normal forms for abstract pure bigraphs have been proposed:

- the *discrete normal form (DNF)* generated from elementary bigraphs and the composition and tensor product; this was presented with a correct and complete axiomatization but turns out to be verbose in practice;
- the *connected normal form (CNF)* that uses parallel composition and wide parallel composition of process algebras; no complete axiomatization for these normal form has been found jet, but this form is more common and easy for "programming" applications.

Each bigraph can be expressed in DNF and CNF uniquely (up to isomorphism) using compositions of elementary bigraphs.

Elementary bigraphs

We often talk of a class generated by certain elements; this means the class formed from those elements using composition and product and identities. In this section we give the elementary bigraphs from which all others can be generated. To avoid too many parentheses in expressions we shall often represent composition by justapoxition; it binds tightly, for example $G_1G_2 \otimes G_3$ means $(G_1 \circ G_2) \otimes G_3$.

We have the placings (bigraphs without edges) 1 (a single region without anything inside), *join* (two holes inside the same region) Π (renaming the holes); the linkings (bigraphs without nodes) w; and the molecule $\mathsf{K}_{\vec{a}}$ (a single region containing a node with names $\vec{a}$ and an hole inside it).

Part II

A New Logic: BiLog

Chapter 3

BiLog framework

Our final aim is to define a logic able to describe bigraphs (and their substructures). As bigraphs, place graphs, and link graphs are arrows of a (partial) monoidal category, we first introduce a meta-logical framework having monoidal categories as models and then we adapt it to model the orthogonal structures of place and link graphs. Finally we specialize the logic to model the whole structure of (abstract) bigraphs.

We follow the approach of spatial logics by introducing connectives that mirror the model structure. In this case models are monoidal categories and the logic spatially describes the structure of the *arrows*[1].

The meta-logical framework proposed in this Chapter is inspired by the bigraph axiomatization presented in [94]. We consider worlds as *terms* of a general language with *horizontal* and *vertical* compositions and a set of unary constructors. We then consider a *structural congruence* on these terms that must satisfy, at least, the axioms of monoidal categories and we provide a model theory that is parametric wrt this structural congruence and the constructors of the considered model. We want to remain as free as possible from the level of intensionality, thus we define the logic depending on a *transparency* predicate whose purpose is to identify which terms allow inspection of their content (*transparent* terms) and which not (*opaque* terms). We inspect the logical equivalence induced by the logic and we observe that it is the structural congruence when the transparency predicate is always true and it is less discriminating when *opaque terms* are present. In addition, we show how some interesting derived connectives as the somewhere modality and assertions constraining the "type" of terms can be easily derived.

In the following chapter we will instantiate this framework to describe bigraphical structures and embed spatial logics presented in Chapter 1.

[1]The logic can be seen as a logic for categories, but we describe the arrows of the category and not the objects as usual for logic for categories (e.g. linear logic).

3.1 BiLog terms

The elements of our logic (i.e., the worlds wrt a formula will be interpreted) are *terms* freely generated from a set of constructors Θ using the operators of composition ($\circ$) and tensor ($\otimes$). These two operations (when defined) must satisfy the bifunctoriality property of monoidal categories, thus we will also refer to these terms as *bifunctorial terms*.

The terms represent structures built on a monoid whose elements are dubbed *interfaces* and denoted by I, J. Since we also want to model nominal resources, like heaps or link graphs, we consider a monoid that can be partial.

Given a set of term constructors Θ, ranged over by Ω, and a partial monoid $(M, \otimes, \epsilon)$, we define BiLog terms (also said bifunctorial terms) in Table 3.1.1.

Table 3.1.1. *BiLog terms*

$G, G' ::=$	BiLog terms
$G \otimes G'$	horizontal composition
$G \circ G'$	vertical composition
Ω	constructor $\Omega \in \Theta$

Intuitively, terms represent typed structures with a source and a target interface ($G : I \to J$). Structures can be placed one near the other (horizontal composition) or one inside (or under) the other (vertical composition). Each Ω in Θ has a type $type(\Omega) = I \to J$. For each interface I, we assume a distinguished construct $id_I : I \to I$. The types of constructors, together with the rules in Table 3.1.2 determine the type of each term. Terms of type $\epsilon \to J$ are called *ground*.

Table 3.1.2. *Typing rules*

$$\frac{type(\Omega) = I \to J}{\Omega : I \to J} \qquad \frac{F : I \to I' \quad G : I' \to J}{G \circ F : I \to J}$$

$$\frac{G : I_1 \to J_1 \quad F : I_2 \to J_2 \quad I = I_1 \otimes I_2 \quad J = J_1 \otimes J_2}{G \otimes F : I \to J}$$

Notice that the tensor term is well typed when both corresponding tensors on source and target interface are defined (i.e., they are separated structures). On the other hand, composition is defined when the two terms *share* a common interface only. From now on we will consider well typed terms only.

We consider terms up to a structural congruence $\equiv$, which subsumes the axioms of monoidal categories 2.1.1. Later on, the congruence will be refined to model specialised structures, such as place graphs or bigraphs. All axioms are required to hold only when both sides are well typed. Here *id* represents any possible identity on whatever interface. Throughout the Thesis, when using $=$ or $\equiv$ we imply that both sides are defined and we write $(G)\downarrow$ to say that G is defined.

Table 3.1.3. *BiLog Congruence Axioms*

Congruence Axioms:	
$G \equiv G'$	Reflexivity
$G \equiv G' \Rightarrow G' \equiv G$	Symmetry
$G \equiv G' \wedge G' \equiv G'' \Rightarrow G \equiv G''$	Transitivity
$G \equiv G' \wedge F \equiv F' \Rightarrow G \circ F \equiv G' \circ F'$	Congruence $\circ$
$G \equiv G' \wedge F \equiv F' \Rightarrow G \otimes F \equiv G' \otimes F'$	Congruence $\otimes$
Axioms of Monoidal Categories:	
$G \circ id \equiv G \equiv id \circ G$	Identity
$(G_1 \circ G_2) \circ G_3 \equiv G_1 \circ (G_2 \circ G_3)$	Associativity
$G \otimes id_\epsilon \equiv G \equiv id_\epsilon \otimes G$	Monoid Neutral Element
$(G_1 \otimes G_2) \otimes G_3 \equiv G_1 \otimes (G_2 \otimes G_3)$	Monoid Associativity
$id_I \otimes id_J \equiv id_{I \otimes J}$	Monoid Identity
$(G_1 \otimes F_1) \circ (G_2 \otimes F_2) \equiv (G_1 \circ G_2) \otimes (F_1 \circ F_2)$	Bifunctoriality

Notice that these axioms correspond to the axioms of (partial) monoidal categories. In particular we constrain the structural congruence to enjoy the bifunctoriality property between the two constructors, namely product and composition. Thus, we can interpret our terms as arrows of the free monoidal category on $(M, \otimes, \epsilon)$ and Θ. In this case the term congruence corresponds to equality of the corresponding arrows.

3.2 The BiLog logic

We are going to define a parametric logical framework for bifunctorial terms in general. When the framework is instantiated, terms specialize to represent particular structures (e.g. place graph or link graph terms) and the logic specializes to describe this kind of terms. A BiLog formula semantics corresponds to a sets of terms. The logic will feature spatial connectives in the sense Spatial Logics introduced in Chapter 1.

3.2.1 Transparency

The fact that a structure exists does not necessarily mean that this structure can be observed in all its details. As an immediate example consider the structure of process terms in a calculus. In this case the structure is used to encode behaviour and not simply to represent the distribution or shape of resources. We may want to avoid a direct representation of these structures in the logic with spatial connectives because the structure is not really spatial. A natural way to solve this is to define a notion of transparency over the structure such that entities whose spatiality represents really the structure are transparent, while entities that encode behavior are opaque and

cannot be distinguished by the logic spatial connectives. Interpreting bifunctorial terms as arrows, transparent terms allow the logic to see all the structure of the arrow till the source interface, while opaque terms block the inspection at some middle point. Observe that the notion of transparency also exists in models without temporal behaviour. Consider as an example a model with an access control policy guided by the structure. The policy could be variable and defined on constructors by the administrator. Thus, some terms may be transparent or opaque depending on the current policy and the visibility in the logic (or query language) will be influenced by this.

When the model is dynamic, reaction rule contexts are specified with an activeness predicate, we may be tempted to identify transparency as the activeness of terms. Even thought these concepts coincide in some case (e.g. usually passive contexts are opaque), they are completely orthogonal in general. We may have transparent terms that are active (e.g. a public location/directory), opaque terms that are active (an agent that hides its content), passive transparent terms (e.g. a script code) and passive opaque terms (e.g. controls encoding synchronization). Indeed, the transparency is *orthogonal* to the concept of activeness.

3.2.2 Syntax and Semantics

BiLog internalises the bifunctorial term constructors in the style of the ambient logic [45]. Constructors are represented in the logic as constant formulas, while tensor product and composition are expressed by connectives. We thus have two binary spatial operators. This contrasts with other spatial logics, which have only one: ambient-like logics, with parallel composition $A \mid B$, Separation Logic [101], with separating conjunction $A * B$, and Context Tree Logic [33], with application $K(P)$. Both our operators inherit the monoidal structure and non-commutativity properties from the model.

Our logic is parametric wrt a transparency predicate τ, reflecting that not every term can be directly observed in the logic: some are opaque and do not allow inspection of their contents. We will see that when all terms are observable (i.e., $\tau(G)$ for all G), logical equivalence corresponds to $\equiv$. Otherwise, it can be less discriminating. We assume that id_I and ground terms (having nothing to hide) are always transparent, and τ preserves $\equiv$, hence $\otimes$ and $\circ$, in particular. The choice of transparency is motivated by the possibility of having a complex structure not always completely visible on the logical level.

Given the monoid $(M, \otimes, \epsilon)$, the set of simple terms Θ, the transparency predicate τ and the structural congruence relation $\equiv$ we formally define the logic BiLog$(M, \otimes, \epsilon, \Theta, \equiv, \tau)$ in Table 3.2.1 and the meaning of formulas is given in terms of a satisfaction relation.

It features a logical constant $\mathbf{\Omega}$ for each *transparent* construct Ω. In particular we have the identity $\mathbf{id}_I$ for each interface I.

Table 3.2.1. *BiLog*$(M, \otimes, \epsilon, \Theta, \equiv, \tau)$

$\mathbf{\Omega} ::=$	$\mathbf{id}_I \mid \ldots$	a constant formula for every Ω s.t. $\tau(\Omega)$	
$A, B ::=$	$\mathbf{F}$	false	
	$A \Rightarrow B$	implication	
	$\mathbf{id}$	identity	
	$\mathbf{\Omega}$	constant for a simple term	
	$A \otimes B$	tensor product	
	$A \circ B$	composition	
	$A \circ\!\!- B$	left comp. adjunct	
	$A -\!\!\circ B$	right comp. adjunct	
	$A \otimes\!\!- B$	left prod. adjunct	
	$A -\!\!\otimes B$	right prod. adjunct	
$G \models \mathbf{F}$	$\stackrel{def}{=}$	never	
$G \models A \Rightarrow B$	$\stackrel{def}{=}$	$G \models A$ implies $G \models B$	
$G \models \mathbf{\Omega}$	$\stackrel{def}{=}$	$G \equiv \Omega$	
$G \models \mathbf{id}$	$\stackrel{def}{=}$	$\exists I.\, G \equiv id_I$	
$G \models A \otimes B$	$\stackrel{def}{=}$	$\exists G_1, G_2.\, G \equiv G_1 \otimes G_2$ and $G_1 \models A$ and $G_2 \models B$	
$G \models A \circ B$	$\stackrel{def}{=}$	$\exists G_1, G_2.\, G \equiv G_1 \circ G_2$ and $\tau(G_1)$ and $G_1 \models A$ and $G_2 \models B$	
$G \models A \circ\!\!- B$	$\stackrel{def}{=}$	$\forall G'.\, G' \models A$ and $\tau(G')$ and $(G' \circ G)\downarrow$ implies $G' \circ G \models B$	
$G \models A -\!\!\circ B$	$\stackrel{def}{=}$	$\tau(G)$ implies $\forall G'.\, G' \models A$ and $(G \circ G')\downarrow$ implies $G \circ G' \models B$	
$G \models A \otimes\!\!- B$	$\stackrel{def}{=}$	$\forall G'.\, G' \models A$ and $(G' \otimes G)\downarrow$ implies $G' \otimes G \models B$	
$G \models A -\!\!\otimes B$	$\stackrel{def}{=}$	$\forall G'.\, G' \models A$ and $(G \otimes G')\downarrow$ implies $G \otimes G' \models B$	

The satisfaction of logical constants is simply the congruence to the corresponding constructor. The *horizontal decomposition* formula $A \otimes B$ is satisfied by a term that can be decomposed as the tensor product of terms satisfying A and B respectively. The degree of separation enforced by $\otimes$ between terms plays a fundamental role in the various instances of the logic (notably link graph and place graph). The *vertical decomposition* formula $A \circ B$ is satisfied by terms that can be seen as the composition of terms satisfying A and B. We shall see that both connectives correspond in some cases to well known spatial connectives. We define the *left* and *right adjuncts* for composition and tensor to express extensional properties. The left adjunct $A \circ\!\!- B$ expresses the property of a term to satisfy B whenever inserted in a context satisfying A. Similarly, the right adjunct $A -\!\!\circ B$ expresses the property of a context to satisfy B whenever filled with a term satisfying A. A similar description for $\otimes\!\!-$ and $-\!\!\otimes$, the adjoints of $\otimes$. Observe that these collapse if the tensor is commutative in the model.

3.2.3 Derived Operators

In Table 3.2.2 we outline some interesting operators that can be derived in BiLog. The operators constraining the interfaces are self-explanatory. The 'dual' operators have the following semantics: $A \ominus B$ is satisfied by terms G such that for every possible decomposition $G_1 \otimes G_2$ either $G_1 \models A$ or $G_2 \models B$. For instance, $A \ominus A$ describes terms where A is true in (at least) one part of each $\otimes$-decomposition. The formula $\mathbf{F} \ominus (\mathbf{T}_{\multimap I} \Rightarrow A) \ominus \mathbf{F}$ describes those terms where every I-component satisfies A. Similarly, the composition $A \bullet B$ expresses structural properties universally quantified on every $\circ$-decomposition.

Both these connectives are useful to specify security properties or types. The adjunct dual $A \bullet\!\!- B$ describes terms that can be inserted into a context satisfying A – a sort of existential quantification on contexts – obtaining a term satisfying B. For instance $(\mathbf{\Omega}_1 \vee \mathbf{\Omega}_2) \bullet\!\!- A$ describes the terms that can be inserted either in Ω_1 or Ω_2 resulting in a term satisfying A. Similarly the adjunct dual $A -\!\!\bullet B$ describes contextual terms G such that there exists a term satisfying A that can be inserted in G to obtain a term satisfying B.

The formulas $A^{\exists\otimes}$, $A^{\forall\otimes}$, $A^{\exists\circ}$, and $A^{\forall\circ}$ correspond to quantifications on the horizontal/vertical structure of terms. For instance $\mathbf{\Omega}^{\forall\circ}$ describes terms that are a finite (possibly empty) composition of simple terms Ω. The equality between interfaces $I = J$ is easily derivable using $\otimes$ and $\otimes\!\!-$.

Deriving somewhere modality

We can extend the idea of *sublocation* ($\sqsubseteq$) defined in [43] to our terms. The inductive definition of $\sqsubseteq$ specifies that $G \sqsubseteq G$, and $G' \sqsubseteq G$ if either $G \equiv G_1 \otimes G_2$, with $G' \sqsubseteq G_1$ (and symmetrically $G' \sqsubseteq G_2$) or $G \equiv G_1 \circ G_2$, with $\tau(G_1)$ and $G' \sqsubseteq G_2$. Exploiting this relation between ground terms, we define a *somewhere* modality. Intuitively, we say that a term satisfies $\diamond A$ whenever one of its sublocations satisfies A. Quite surprisingly, $\diamond A$ is expressible in the logic.

We follow the definition of sublocation in [43], and we extend the idea by considering all the term constructors.

Definition 3.2.1 (Sublocation)**.** *Given two terms $G : \epsilon \to J$ and $G' : \epsilon \to J'$, we define G' to be a* sublocation *for G, and we write $G' \sqsubseteq G$, inductively by:*

- *$G' \sqsubseteq G$, if $G' \equiv G$;*
- *$G' \sqsubseteq G$, if $G \equiv G_1 \otimes G_2$, with $G' \sqsubseteq G_1$ or $G' \sqsubseteq G_2$;*
- *$G' \sqsubseteq G$, if $G \equiv G_1 \circ G_2$, with $\tau(G_1)$ and $G' \sqsubseteq G_2$.*

The definition of a "somewhere" modality makes sense only for terms of type $\epsilon \to J$. The usual way to define the semantics for the "somewhere" modality is to define that a term $G : \epsilon \to J$ satisfies the formula "*somewhere*" A if and only if

$$\text{there exists } G' \sqsubseteq G \text{ such that } G' \models A.$$

Table 3.2.2. *Derived Operators*

$\mathbf{T}, \wedge, \vee, \Leftrightarrow, \Leftarrow, \neg$		Classical operators
A_I	$\stackrel{def}{=} A \circ \mathbf{id}_I$	Constraining the source to be I
$A_{\rightarrow J}$	$\stackrel{def}{=} \mathbf{id}_J \circ A$	Constraining the target to be J
$A_{I \rightarrow J}$	$\stackrel{def}{=} (A_I)_{\rightarrow J}$	Constraining the type to be $I \rightarrow J$
$A \circ_I B$	$\stackrel{def}{=} A \circ \mathbf{id}_I \circ B$	Composition with interface I
$A \circ\!\!- _J B$	$\stackrel{def}{=} A_{\rightarrow J} \circ\!\!- B$	Contexts with J as target guarantee
$A \multimap_I B$	$\stackrel{def}{=} A_I \multimap B$	Guarantee on terms with I as source
$A \ominus B$	$\stackrel{def}{=} \neg(\neg A \otimes \neg B)$	Dual of tensor product
$A \bullet B$	$\stackrel{def}{=} \neg(\neg A \circ \neg B)$	Dual of composition
$A \bullet - B$	$\stackrel{def}{=} \neg(\neg A \circ\!\!- \neg B)$	Dual of composition left adjunct
$A - \!\!\bullet B$	$\stackrel{def}{=} \neg(\neg A \multimap \neg B)$	Dual of composition right adjunct
$A^{\exists \otimes}$	$\stackrel{def}{=} \mathbf{T} \otimes A \otimes \mathbf{T}$	Some horizontal term satisfies A
$A^{\forall \otimes}$	$\stackrel{def}{=} \mathbf{F} \ominus A \ominus \mathbf{F}$	Every horizontal term satisfies A
$A^{\exists \circ}$	$\stackrel{def}{=} \mathbf{T} \circ A \circ \mathbf{T}$	Some vertical term satisfies A
$A^{\forall \circ}$	$\stackrel{def}{=} \mathbf{F} \bullet A \bullet \mathbf{F}$	Every vertical term satisfies A
$I = J$	$\stackrel{def}{=} \mathbf{T} \otimes (id_\epsilon \wedge id_I \otimes\!\!- id_J)$	Equality between interfaces
$\diamondsuit A$	$\stackrel{def}{=} (\mathbf{T} \circ A)_\epsilon$	Somewhere modality (on ground terms)
$\boxplus A$	$\stackrel{def}{=} \neg\diamondsuit\neg A$	Anywhere modality (on ground terms)

In the case of terms typed by $\epsilon \rightarrow J$, the previous requirement is the semantics of the connective $\diamondsuit$ as defined in Table 3.2.2.

Proposition 3.2.2. *For every term G of type $\epsilon \rightarrow J$, it is the case that*

$$G \models \diamondsuit A \text{ if and only if there exists } G' \sqsubseteq G \text{ such that } G' \models A.$$

Proof. First prove a supporting property characterising the relation between a term and its sublocations.

Property 3.2.3. *For every term $G : \epsilon \rightarrow J$ and $G' : \epsilon \rightarrow J'$, we have: $G' \sqsubseteq G$ if and only if there exists a term C such that $\tau(C)$ and $G \equiv C \circ G'$.*

The direction from right to left is a simple application of Definition 3.2.1. The direction from left to right is proved by induction on Definition 3.2.1. For the *basic step*, the implication clearly holds if $G' \sqsubseteq G$ in case $G' \equiv G$. In the *inductive step* we distinguish two cases.

1. Suppose $G' \sqsubseteq G$ is due to the fact that $G \equiv G_1 \otimes G_2$, with $G' \sqsubseteq G_1$ or $G' \sqsubseteq G_2$. Without loss of generality, assume $G' \sqsubseteq G_1$. The induction says that there exists C such that $\tau(C)$ and $G_1 \equiv C \circ G'$. Hence, $G \equiv (C \circ G') \otimes G_2$. Now the typing is:

$$C : I_C \rightarrow J_C \qquad G' : \epsilon \rightarrow I_C \qquad G_2 : \epsilon \rightarrow J_2 \qquad G : \epsilon \otimes \epsilon \rightarrow J_C \otimes J_2,$$

so $G \equiv (C \circ G') \otimes (G_2 \circ id_\epsilon)$. As the interface ϵ is the neutral element for the tensor product between interfaces, compose

$$C \otimes G_2 : I_C \otimes \epsilon \to J_C \otimes J_2 \qquad G' \otimes id_\epsilon : \epsilon \otimes \epsilon \to I_C \otimes \epsilon$$

and hence the term $(C \otimes G_2) \circ (G' \otimes id_\epsilon)$ is defined. Note that $\tau(C \otimes G_2)$ is verified, in fact, $\tau(G_2)$ is verified as $G_2 : \epsilon \to J_2$ and $\tau(C)$ is verified by induction. Hence, by bifunctoriality property, conclude $G \equiv (C \otimes G_2) \circ G'$, with $\tau(C \otimes G_2)$, as aimed.

2. Suppose $G' \sqsubseteq G$ is due to the fact that $G \equiv G_1 \circ G_2$, with $\tau(G_1)$ and $G' \sqsubseteq G_2$. The induction says that there exists C such that $\tau(C)$ and $G_2 \equiv C \circ G'$. Hence, $G \equiv G_1 \circ (C \circ G')$. Conclude $G \equiv (G_1 \circ C) \circ G'$, with $\tau(G_1 \circ C)$.

Suppose now that $G \models \Diamond A$, this means that $G \models (\mathbf{T} \circ A)_\epsilon$. According to Tab. 3.1, this means that there exist C and G' such that $G' \models A$ and $\tau(C)$, and $G \equiv C \circ G'$. Finally, by Property 3.2.3, this means $G' \sqsubseteq G$ and $G' \models A$. □

The *everywhere* modality ($\boxminus$) is dual to $\Diamond$. A term satisfies the formula $\boxminus A$ if each of its sublocations satisfies A.

3.2.4 Logical equivalence and transparency

We now show some basic results about BiLog and some of its instances. In particular, we observe that, in presence of trivial transparency, the induced logical equivalences coincide with the structural congruences for the terms they describe. This property is fundamental in order to describe, query and reason about bigraphical data structures, as e.g. XML (cf. [60]). In other terms, BiLog is *intensional* in the sense of [107] (i.e., it can observe internal structures), as opposed to the extensional logics used to observe the behaviour of dynamic system. Following [74], it would be possible to study a fragment of BiLog without intensional operators ($\otimes$, $\circ$, and constants).

The lemma below states that the relation $\models$ is well defined w.r.t. the congruence and that the interfaces for transparent terms can be observed.

Lemma 3.2.4 (Type and Congruence preservation)**.** *For every couple of term G, G', it holds: $G \models A$ and $G \equiv G'$ implies $G' \models A$.*
For every term G, it holds: $G \models A_{I \to J}$ if and only if $G : I \to J$, $G \models A$, and $\tau(G)$.

Proof. We prove the first point by induction on the structure of the formula, by recalling that a congruence need to preserve the typing and the transparency (i.e., if $G \equiv G'$ then $\tau(G)$ if and only if $\tau(G')$). In detail we have

Case F Nothing to prove.

Case $A \Rightarrow B$ By hyp. $G \models A \Rightarrow B$ and $G \equiv G'$. This means that if $G \models A$ then $G \models B$. By induction if $G' \models A$ then $G \models A$. Thus if $G' \models A$ then $G \models B$ and by induction $G' \models B$.

Case $\mathbf{\Omega}$ By hyp. $G \models \mathbf{\Omega}$ and $G \equiv G'$. By definition of satisfaction $G \equiv \Omega$ and by transitivity of congruence $G' \equiv \Omega$ and thus $G' \models \mathbf{\Omega}$.

Case id By hyp. $G \models \mathbf{id}$ and $G \equiv G'$. Thus there exists an I such that $G \equiv G' \equiv id_I$ and so $G' \models \mathbf{id}$.

Case $A \otimes B$ By hyp. $G \models A \otimes B$ and $G \equiv G'$. Thus there exist G_1, G_2 such that $G \equiv G' \equiv G_1 \otimes G_2$ and $G_1 \models A$ and $G_2 \models B$. Thus $G' \models A \otimes B$.

Case $A \circ B$ By hyp. $G \models A \circ B$ and $G \equiv G'$. Thus there exist G_1, G_2 such that $G \equiv G' \equiv G_1 \circ G_2$, $\tau(G_1)$ and $G_1 \models A$ and $G_2 \models B$. Thus $G' \models A \circ B$.

Case $A \circ\!\!- B$ By hyp. $G \models A \circ\!\!- B$ and $G \equiv G'$. Thus for each G'' such that $G'' \models A$ and $\tau(G'')$ and $(G'' \circ G)\downarrow$ then $G'' \circ G \models B$. Now by congruence $G \equiv G'$ implies $G'' \circ G \equiv G'' \circ G'$, we recall that congruence must preserve typing so $(G'' \circ G')\downarrow$. Thus by induction $G'' \circ G' \models B$ and this is true for all the considered G'', thus we have proved that $G' \models A \circ\!\!- B$.

Case $A \multimap B$ If $\tau(G')$ is false $G' \models A \multimap B$ trivially holds. So suppose $\tau(G')$, since $G \equiv G'$ and transparency preserve congruence we have that $\tau(G)$. By hyp. for each G'' satisfying A such that $(G \circ G'')\downarrow$ we have that $G \circ G'' \models B$, and by induction $G' \circ G'' \models B$ (again $G \equiv G'$ and $(G \circ G'')\downarrow$ implies $(G' \circ G'')\downarrow$). This proves $G' \models A \multimap B$

Case $A \otimes\!\!- B$ (and symmetrically $A \multimap\!\!\otimes B$). By hyp. $G \models A \otimes\!\!- B$ and $G \equiv G'$. Thus for each G'' such that $G'' \models A$ and $(G'' \otimes G)\downarrow$ then $G'' \otimes G \models B$. Now by congruence $G \equiv G'$ implies $G'' \otimes G \equiv G'' \otimes G'$, again the congruence must preserve typing so $(G'' \otimes G')\downarrow$. Thus by induction $G'' \otimes G' \models B$ and this is true for all the G'' considered, thus we have proved that $G' \models A \otimes\!\!- B$.

The second point makes use of the first one. For the forward direction, assume that $G \models A_{I \to J}$, then $G \equiv id_J \circ G' \circ id_I$ with $G' \models A$ and $\tau(G')$. Now, $id_J \circ G' \circ id_I : I \to J$. Thanks to the first point we obtain $G : I \to J$, $G \models A$ and $\tau(G)$. The converse is a direct consequence of the definition for the satisfaction of formulas. □

BiLog induces a logical equivalence $=_L$ on terms in the usual sense, that is $G_1 =_L G_2$ if $G_1 \models A$ implies $G_2 \models A$ and vice versa, for every formula A. It is easy to prove that the logical equivalence corresponds to the congruence in the model, if the transparency predicate is total, i.e., $\tau(G)$ for every G.

Theorem 3.2.5 (Logical equivalence and congruence)**.** *If the transparency predicate is always true, then for every terms G, G': $G =_L G'$ if and only if $G \equiv G'$.*

Proof. The forward direction is proved by defining the characteristic formula of terms. Please note that every term can be expressed as a formula, in fact every constant term presents a corresponding constant formula, since the transparency predicate is total. The converse is a direct consequence of Lemma 3.2.4. □

The particular characterization of logical equivalence as the congruence in the case of trivial transparency can be generalized to the congruence up-to-transparency. That means we can find an equivalence relation between trees that is tuned by τ, more τ covers, less the equivalence distinguishes.

To do this, however, we must take care of the way transparency is defined. Our idea of transparency (and opaqueness) is essentially a way to restrict the observational power of the logic in the current state (i.e., in the static logic). Notice that a restriction of the observational power in the static logic does not hinder in general a restriction of the observational power in the dynamic counterpart, that is because the next step modality could allow a re-intensionalization of the controls by observing how the model evolve (how it is shown in Sangiorgi and Lozes works).

In general, not every structure of the model corresponds to an observable structure in the spatial logic. A classical example is in the ambient logic. Some mobile ambient constructors have their logical equivalent (e.g. ambients), other constructors are not directly mapped in the logic (e.g. **in** and **out** prefixes). In this case the observability of the structure is distinguished from the observability of the computational terms, i.e., some terms are used to express behavior and other to express structure. Ambients also have terms representing both structure and possible behavior (since ambients can be opened).

However the transparency of controls is not necessarily related to the dynamic behavior (thus we must distinguish between transparent terms and active terms). For example, we may have two BiLog logics on the same terms with two different level of transparency. In this case some controls are sealed for the first logic and open for the second one. An example could be directories in a file system with a two-level access policy. The administrator could have access to all the directories (i.e., transparency always true for its logic) and the user cannot access some privileged directories (i.e., the opaque controls for him), he could also ignore the possibility to have this kind of directory (i.e., their formula is not present in its logic).

More generally the transparency predicate is needed in order to avoid that every single term in the structure is mapped to its logical equivalent. Models can have additional structure that is not observable. As an other example consider an XML document. We may want to consider the content only of some kind of nodes, for example we could ignore data values as their addition in the logic could add complexity, or because we want to look only at the structure. On the other hand another logic could be interested in values, but not in attributes of the nodes. The point is that we can avoid an encoding of models into easier models simply by avoiding model inspection inside the undesired controls.

In the logic we gave the minimal restrictions on the transparency predicate to

prove our results. From now on we study transparency in more detail. The most natural way to define the transparency is by making the transparent terms a subcategory of the more general category of terms. This essentially means imposing product and composition of two transparent terms to be transparent.

Thus transparency on all terms can be defined as derived from a transparency policy τ_Θ defined on the constructors only. Note that the transparency definition depends also on the congruence $\equiv$ we are using. In the following we show how the transparency can be derived from a transparency policy

Definition 3.2.6 (Transparency)**.** *Given the monoid of interfaces $(M, \otimes, \epsilon)$, the set of constructors Θ, the congruence $\equiv$ and a transparency policy predicate τ_Θ defined on the constructors in Θ we define the transparency on terms as follows:*

$$\frac{G \equiv id_I}{\tau(G)} \qquad \frac{\exists I.G : \epsilon \to I}{\tau(G)} \qquad \frac{G \equiv \Omega \quad \tau_\Theta(\Omega)}{\tau(G)}$$

$$\frac{G \equiv G_1 \otimes G_2 \quad \tau(G_1) \quad \tau(G_2)}{\tau(G)} \qquad \frac{G \equiv G_1 \circ G_2 \quad \tau(G_1) \quad \tau(G_2)}{\tau(G)}$$

Now we have to prove that the condition we posed on the transparency predicate holds for this particular definition

Lemma 3.2.7 (Transparency properties)**.** *If G is ground or G is an identity then $\tau(G)$ is true;*
If $G \equiv G'$ then $\tau(G)$ is equivalent to $\tau(G')$.

Proof. The first are by definition. The second can be easily proved by induction on the derivations. □

3.2.5 Logical properties

Notice that for every axiom of the model we can prove a corresponding logical property. In particular the bifunctoriality property is expressed by formulas $(A_I \circ B_{\to I}) \otimes (A'_J \circ B'_{\to J}) \Leftrightarrow (A_I \otimes A'_J) \circ (B_{\to I} \otimes B'_{\to J})$ that are valid when $(I \otimes J)\downarrow$.

In general, given two formulas A, B we say that A *satisfies* B, and we write $A \vdash B$, if for every term G it is the case that $G \models A$ implies $G \models B$.

Assume that I and J are two interfaces such that their tensor product $I \otimes J$ is defined. Then, the bifuctoriality property in the logic is expressed by[2]

$$(A_I \circ B_{\to I}) \otimes (A'_J \circ B'_{\to J}) \dashv\vdash (A_I \otimes A'_J) \circ (B_{\to I} \otimes B'_{\to J}). \tag{3.1}$$

In fact, we prove the following

Proposition 3.2.8. *The equation (3.1) holds in the logic whenever $(I \otimes J)\downarrow$.*

[2]We generalise the satisfaction between formulas, and we write $A \dashv\vdash B$ to say both $A \vdash B$ and $B \vdash A$.

Proof. We prove separately the two way of the satisfaction. First we prove

$$(A_I \circ B_{\to I}) \otimes (A'_J \circ B'_{\to J}) \vdash (A_I \otimes A'_J) \circ (B_{\to I} \otimes B'_{\to J}) \tag{3.2}$$

Assume that $G \models (A_I \circ B_{\to I}) \otimes (A'_J \circ B'_{\to J})$. This means that there exist $G' : I' \to I''$, $G'' : J' \to J''$ such that $I' \otimes J'$ and $I'' \otimes J''$ are defined, and $G \equiv G' \otimes G''$, with $G' \models A_I \circ B_{\to I}$ and $G'' \models A'_J \circ B'_{\to J}$. Now, $G' \models A_I \circ B_{\to I}$ means that there exists $G_1 : I_1 \to J_1$, $G_2 : I_2 \to I_1$ such that: $\tau(G_1)$, $G' \equiv G_1 \circ G_2$, $G_1 \models A_I$, and $G_2 \models B_{\to I}$. Thus, we deduce that $G' \equiv G_1 \circ G_2$, with:

- $G_1 : I \to J'$ such that $\tau(G_1)$, and $G_1 \models A$;
- $G_2 : I' \to I$ such that $G_2 \models B$.

Similarly, we prove that $G'' \equiv G'_1 \circ G'_2$, with:

- $G'_1 : J \to J''$ such that $\tau(G'_1)$, and $G'_1 \models A'$;
- $G'_2 : I'' \to J$ such that $G_2 \models B'$.

In particular, we obtain $G \equiv (G_1 \circ G_2) \otimes (G'_1 \circ G'_2)$. Since we can perform the tensor product of the required interfaces (please recall that $I \otimes J$ is defined), we can compose $(G_1 \otimes G'_1) \circ (G_2 \otimes G'_2)$. The bifunctoriality property implies that $G \equiv (G_1 \otimes G'_1) \circ (G_2 \otimes G'_2)$. Moreover we have $\tau(G_1 \otimes G'_1)$, as $\tau(G_1)$ and $\tau(G'_1)$. Hence we conclude that $G \models (A_I \otimes A'_J) \circ (B_{\to I} \otimes B'_{\to J})$, as required.

For the converse, we have to prove

$$(A_I \otimes A'_J) \circ (B_{\to I} \otimes B'_{\to J}) \vdash (A_I \circ B_{\to I}) \otimes (A'_J \circ B'_{\to J}). \tag{3.3}$$

Assume that $G \models (A_I \otimes A'_J) \circ (B_{\to I} \otimes B'_{\to J})$. By following the same lines as for (3.2), we deduce that $G \equiv (G_1 \otimes G'_1) \circ (G_2 \otimes G'_2)$, where

- $\tau(G_1 \otimes G'_1)$;
- $G_1 : I \to J'$ such that $G_1 \models A$;
- $G'_1 : J \to J''$ such that $G'_1 \models A'$;
- $G_2 : I' \to I$ such that $G_2 \models B$.
- $G'_2 : I'' \to J$ such that $G_2 \models B'$.

Also in this case, we can perform the tensor product of the required interfaces. Hence we can compose $(G_1 \circ G_2) \otimes (G'_1 \circ G'_2)$. Again, bifunctoriality property implies $G \equiv (G_1 \circ G_2) \otimes (G'_1 \circ G'_2)$. Finally, by observing that $\tau(G_1 \otimes G'_1)$ implies $\tau(G_1)$ and $\tau(G'_1)$, we can deduce $G_1 \circ G_2 \models (A_I \circ B_{\to I})$ and $(G'_1 \circ G'_2) \models (A'_J \circ B'_{\to J})$. Then we conclude $G \models (A_I \circ B_{\to I}) \otimes (A'_J \circ B'_{\to J})$.

□

Chapter 4

BiLog instances

In this chapter we instantiate the BiLog framework to describe place graphs, link graphs, and bigraphs respectively. We obtain a spatial logic for bigraphs as a natural composition of a place graph logic (for tree contexts) and a link graph logic (for name linkings). For each logic instance we prove an embedding result of a spatial logic presented in the overview of Chapter 1.

4.1 A Logic for distributed resources

The main point is that a resource has a spatial structure as well as a link structure associated to it. Suppose for instance to be describing a tree-shaped distribution of resources in locations. We may use atomic formulas like $\mathsf{PC}(A)$ and $\mathsf{PC}_x(A)$ to describe a resource in an unnamed location, respectively location x, of 'type' PC (e.g. a computer) whose contents satisfy A. Note that the location type is orthogonal to the name. We can then write $\mathsf{PC}(\mathbf{T}) \otimes \mathsf{PC}(\mathbf{T})$ to characterise terms with two unnamed PC resources whose contents satisfy the tautological formula (i.e., with anything inside). Using named locations, as e.g. in $\mathsf{PC}_a(\mathbf{T}) \otimes \mathsf{PC}_b(\mathbf{T})$, we are able to express name separation, i.e., that names a and b are different. Furthermore, using link expressions we can force name-sharing between resources with formulas like:

$$\mathsf{PC}_a(\mathsf{in}_c \otimes \mathbf{T}) \overset{c}{\otimes} \mathsf{PC}_b(\mathsf{out}_c \otimes \mathbf{T})$$

This describes two PC with different names, a and b, sharing a link on a distinct name c, which models, e.g., a communication channel. Name c is used as input (in) for the first PC and as an output (out) for the second PC. No other names are shared and c cannot be used elsewhere inside the PCs.

A bigraphical structure is, in general, a context with several holes and open links that can be filled by composition. This means that the logic can describe contexts for resources at no additional cost. We can then express formulas like $\mathsf{PC}_a(\mathbf{T} \otimes \mathsf{HD}(id_1))$ that describes a modular computer PC, where id_1 represents a 'pluggable' hole in the hard disc HD. Contextual resources have many important

applications. In particular, the contextuality of bigraphs is useful to specify reaction rules, but it can also be used as a general mechanism to describe contexts of bigraphical data structures (cf. [60, 73]).

As bigraphs are establishing themselves as a truly general (meta)model of global systems, and appear to encompass several existing calculi and models (cf. [80, 95]), our bigraph logic, *BiLog*, aims at achieving the same generality as a description language: as bigraphs specialise to particular models, we expect BiLog to specialise to powerful logics on these. In this sense, our contribution is to propose BiLog as a unifying language for the description of global resources. We will explore this path in future work, fortified by the positive preliminary results obtained for semistructured data [60] and CCS [59].

4.2 Place Graph Logic

Place graphs are essentially ordered lists of regions hosting unordered labelled trees with holes. The labels of the trees correspond to controls K belonging to the fixed signature $\mathcal{K}$. We consider the monoid $(\omega, +, 0)$ of finite ordinals m, n. Interfaces here represent the number of holes and regions of place graphs. Place graph terms are generated from the set $\Theta = \{1 : 0 \to 1, id_n : n \to n, join : 2 \to 1, \gamma_{m,n} : m + n \to n + m, \mathsf{K} : 1 \to 1 \text{ for } \mathsf{K} \in \mathcal{K}\}$. The main structural term is K, that represents a region containing a single node with a hole inside. Other simple terms are *placings*, representing trees $m \to n$ with no nodes; The place identity id_n is neutral for composition. The constructor 1 represents a barren region; *join* is a mapping of two regions into one; $\gamma_{m,n}$ is a permutation that interchanges the first m regions with the following n. The structural congruence $\equiv$ for place graph terms is refined by the usual axioms for symmetry of $\gamma_{m,n}$ and by the place axioms that essentially turn the operation $join \circ (\_ \otimes \_)$ in a commutative monoid with neutral element 1. Hence, the places generated by composition and tensor product from $\gamma_{m,n}$ are *permutations*. A place graph is *prime* if it has type $I \to 1$ (i.e., with a single region).

Table 4.2.1. *Additional Axioms for Place Graphs Structural Congruence*

Symmetric Category Axioms:	
$\gamma_{m,0} \equiv id_m$	Symmetry Id
$\gamma_{m,n} \circ \gamma_{n,m} \equiv id_{m \otimes n}$	Symmetry Composition
$\gamma_{m',n'} \circ (G \otimes F) \equiv (F \otimes G) \circ \gamma_{m,n}$	Symmetry Monoid
Place Axioms:	
$join \circ (1 \otimes id_1) \equiv id_1$	Unit
$join \circ (join \otimes id_1) \equiv join \circ (id_1 \otimes join)$	Associativity
$join \circ \gamma_{1,1} \equiv join$	Commutativity

Example 4.2.1. *service* $\circ$ (*join* $\circ$ (*name* $\otimes$ *description*)) $\otimes$ *push* $\circ$ 1 *is a place graph term on the signature containing* $\{service, name, description, push\}$. *It represents an ordered pair of trees. The first tree is labelled service and has name and description as (unordered) children, both children are actually holed contexts. The second tree is ground having a single node without children. The term has type* $: 2 \to 2$ *and is congruent to* (*service* $\otimes$ *push*) $\circ$ (*join* $\otimes$ 1) $\circ$ (*description* $\otimes$ *name*) $\circ$ id_2. *This contextual pair of trees can be interpreted as semi-structured partially completed data (e.g. an XML message, a web service descriptor) that can be filled by means of composition. Notice that, even if the order between children of the same node is not modeled, the order is still important for composition of contexts with several holes. For instance* $(\mathsf{K}_1 \otimes \mathsf{K}_2) \circ (\mathsf{K}_3 \otimes 1)$ *is different from* $(\mathsf{K}_1 \otimes \mathsf{K}_2) \circ (1 \otimes \mathsf{K}_3)$, *as node* K_3 *goes inside* K_2 *in the first case, and inside* K_2 *in the second one.*

Defined the transparency predicate τ on each control in $\mathcal{K}$, the Place Graph Logic PGL$(\mathcal{K}, \tau)$ is $BiLog(\omega, +, 0, \equiv, \mathcal{K} \cup \{1, join, \gamma_{m,n}\}, \tau)$. We assume the τ to be true for *join* and $\gamma_{m,n}$. It follows from Theorem 3.2.5 that PGL can describe place graphs precisely. The logic resembles a propositional spatial tree logic, like [32]. The main differences are that PGL models contexts of trees and that the tensor product is not commutative, unlike the parallel composition, allowing us to model the order of regions. We can define a commutative separation using **join** and the tensor product, the *parallel composition* $A \mid B \stackrel{def}{=}$ **join** $\circ$ $(A_{\to 1} \otimes B_{\to 1})$. This separation is purely structural, and corresponds at term level to *join* $\circ$ $(P \otimes P')$ that is a total operation on all prime place graphs.

4.2.1 Encoding STL

Not surprisingly, prime (single-region) ground place graphs are isomorphic to the unordered trees that models the static fragment of ambient. We show that BiLog restricted to prime ground place graphs (with the always-true transparency predicate) is equivalent to the propositional spatial tree logic of [32] (STL in the following). The logic STL expresses properties of unordered labelled trees T constructed from the empty tree 0, the labelled node containing a tree $a[T]$, and the parallel composition of trees $T_1 \mid T_2$. It is a static fragment of the ambient logic [45] characterized by propositional connectives, spatial connectives (i.e., 0, $a[A]$, $A \mid B$), and their adjuncts (i.e., $A@a$, $A \triangleright B$).

In Table 4.2.2 we encode the tree model of STL into prime ground place graphs, and STL operators into PGL operators. We assume a bijective encoding between labels and controls, and associate every label a with a distinct control $\mathsf{K}(a)$. The monoidal properties of parallel composition are guaranteed by the symmetry and unit axioms of *join*. The equations are self-explanatory once we remark that: *(i)* the parallel composition of STL is the structural commutative separation of PGL; *(ii)* tree labels can be represented by the corresponding controls of the place graph; and *(iii)* location and composition adjuncts of STL are encoded in terms of the left

Table 4.2.2. *Encoding STL in PGL over prime ground place graphs*

Trees into Prime Ground Place Graphs		
$[\![\, 0 \,]\!] \stackrel{def}{=} 1$	$[\![\, a[T] \,]\!] \stackrel{def}{=} \mathsf{K}(a) \circ [\![\, T \,]\!]$	$[\![\, T_1 \mid T_2 \,]\!] \stackrel{def}{=} join \circ ([\![\, T_1 \,]\!] \otimes [\![\, T_2 \,]\!])$
STL formulas into PGL formulas		
$[\![\, \mathbf{0} \,]\!] \stackrel{def}{=} 1$		$[\![\, a[A] \,]\!] \stackrel{def}{=} \mathsf{K}(a) \circ_1 [\![\, A \,]\!]$
$[\![\, \mathbf{F} \,]\!] \stackrel{def}{=} \mathbf{F}$		$[\![\, A@a \,]\!] \stackrel{def}{=} \mathsf{K}(a) \circ\!\!-_1 [\![\, A \,]\!]$
$[\![\, A \Rightarrow B \,]\!] \stackrel{def}{=} [\![\, A \,]\!] \Rightarrow [\![\, B \,]\!]$		$[\![\, A \mid B \,]\!] \stackrel{def}{=} [\![\, A \,]\!] \mid [\![\, B \,]\!]$
$[\![\, A \triangleright B \,]\!] \stackrel{def}{=} ([\![\, A \,]\!] \mid \mathbf{id}_1) \circ\!\!-_1 [\![\, B \,]\!]$		

composition adjunct, as they add logically expressible contexts to the tree. This encoding allows us to prove the following.

The theorem of discrete normal form in [94] implies that every ground place graph $g : 0 \to 1$ may be expressed as

$$g = join_n \circ (M_0 \otimes \ldots \otimes M_{n-1}) \tag{4.1}$$

where every M_j is a molecular prime ground place graph of the form

$$M = \mathsf{K}(a) \circ g,$$

with $ar(\mathsf{K}(a)) = 0$. As an auxiliary notation, $join_n$ is inductively defined as

$$\begin{array}{rcl} join_0 & \stackrel{def}{=} & 1 \\ join_{n+1} & \stackrel{def}{=} & join \circ (id_1 \otimes join_n) \end{array}$$

The theorem in [94] says that the normal form definded in (4.1) is unique, modulo permutations.

For every prime ground place graph, the inverse encoding $(\![\ ]\!)$ considers its discrete normal form and it is inductively defined as follows

$$\begin{array}{rcl} (\![\, join_0 \,]\!) & \stackrel{def}{=} & 0 \\ (\![\, \mathsf{K}(a) \circ q \,]\!) & \stackrel{def}{=} & a[\,(\![\, q \,]\!)\,] \\ (\![\, join_s \circ (M_0 \otimes \ldots \otimes M_{s-1}) \,]\!) & \stackrel{def}{=} & (\![\, M_0 \,]\!) \mid \ldots \mid (\![\, M_{s-1} \,]\!) \end{array}$$

By noticing that the bifunctoriality property implies

$$\begin{array}{l} join_n \circ (M_0 \otimes \ldots \otimes M_{n-1}) \equiv \\ \quad \equiv join \circ (M_0 \otimes (join \circ (M_1 \otimes (join \circ (\ldots \otimes (join \circ (M_{n-2} \otimes M_{n-1}))))))), \end{array}$$

it is easy to see that the encodings $[\![\ ]\!]$ and $(\![\ ]\!)$ are one the inverse of the other, hence they give a bijection from trees to prime ground place graphs, fundamental in the proof of the following theorem.

Theorem 4.2.2 (Encoding STL)**.** *For each tree T and formula A of STL:*

$$T \models_{\text{STL}} A \quad \textit{if and only if} \quad [\![T]\!] \models [\![A]\!].$$

Proof. The theorem is proved by structural induction on STL formulae. The transparency predicate is not considered here, as it is verified on every control. The basic step deals with the constants **F** and **0**. Case **F** follows by definition. For the case **0**, $[\![T]\!] \models [\![\mathbf{0}]\!]$ means $[\![T]\!] \models 1$, that by definition is $[\![T]\!] \equiv 1$ and so $T \equiv (\!| [\![T]\!] |\!) \equiv (\!| 1 |\!) \stackrel{def}{=} 0$, namely $T \models_{\text{STL}} \mathbf{0}$.

The inductive steps deal with connectives and modalities.

Case $A \Rightarrow B$**.** Assuming $[\![T]\!] \models [\![A \Rightarrow B]\!]$ means $[\![T]\!] \models [\![A]\!] \Rightarrow [\![B]\!]$; by definition this says that $[\![T]\!] \models [\![A]\!]$ implies $[\![T]\!] \models [\![B]\!]$. By induction hypothesis, this is equivalent to say that $T \models_{\text{STL}} A$ implies $T \models_{\text{STL}} B$, namely $T \models_{\text{STL}} A \Rightarrow B$.

Case $a[A]$**.** Assuming $[\![T]\!] \models [\![a[A]]\!]$ means $[\![T]\!] \models \mathsf{K}(a) \circ_1 ([\![A]\!])$. This amount to say that there exist $G : 1 \to 1$ and $g : 0 \to 1$ such that $[\![T]\!] \equiv G \circ g$ and $G \models \mathsf{K}(a)$ and $g \models [\![A]\!]$, that is $[\![T]\!] \equiv \mathsf{K}(a) \circ g$ with $g \models [\![A]\!]$. Since the encoding is bijective, this is equivalent to $T \equiv (\!| \mathsf{K}(a) \circ g |\!) \stackrel{def}{=} a[(\!| g |\!)]$ with $g \models [\![A]\!]$. Since $g : 0 \to 1$, the induction hypothesis says that $(\!| g |\!) \models A$. Hence it is the case that $T \models_{\text{STL}} a[A]$.

Case $A@a$**.** Assuming $[\![T]\!] \models [\![A@a]\!]$ means $[\![T]\!] \models \mathsf{K}(a) \multimap_1 A$. This is equivalent to say that for every G such that $G \models \mathsf{K}(a)$, if $(G \circ [\![T]\!])\!\downarrow$ then $G \circ [\![T]\!] \models [\![A]\!]$. According to the definitions, this is $\mathsf{K}(a) \circ [\![T]\!] \models [\![A]\!]$, and so $[\![a[T]]\!] \models [\![A]\!]$. By induction hypothesis, this is $a[T] \models_{\text{STL}} A$. Hence $T \models_{\text{STL}} A@a$ by definition.

Case $A \mid B$**.** Assuming that $[\![T]\!] \models [\![A \mid B]\!]$ means $[\![T]\!] \models [\![A]\!] \mid [\![B]\!]$. This is equivalent to say that $[\![T]\!] \models \mathbf{join} \circ ([\![A]\!]_{\to 1} \otimes [\![B]\!]_{\to 1})$, namely there exist $g_1, g_2 : 0 \to 1$ such that $[\![T]\!] \equiv \mathit{join} \circ (g_1 \otimes g_2)$ and $g_1 \models [\![A]\!]$ and $g_2 \models [\![B]\!]$. As the encoding is bijective this means that $T \equiv (\!| g_1 |\!) \mid (\!| g_2 |\!)$, and the induction hypothesis says that $(\!| g_1 |\!) \models A$ and $(\!| g_2 |\!) \models B$. By definition this is $T \models_{\text{STL}} A \mid B$.

Case $A \triangleright B$**.** Assuming that $[\![T]\!] \models [\![A \triangleright B]\!]$ means

$$[\![T]\!] \models \mathbf{join}([\![A]\!] \otimes \mathbf{id}_1)) \multimap_1 [\![B]\!]$$

namely, for every $G : 1 \to 1$ such that $G \models \mathbf{join}([\![A]\!] \otimes \mathbf{id}_1)$ it holds $G \circ [\![T]\!] \models [\![B]\!]$. Now, $G : 1 \to 1$ and $G \models \mathbf{join}([\![A]\!] \otimes \mathbf{id}_1)$ means that there exists $g : 0 \to 1$ such that $g \models [\![A]\!]$ and $G \equiv \mathit{join}(g \otimes \mathit{id}_1)$. Hence it is the case that for every $g : 0 \to 1$ such that $g \models [\![A]\!]$ it holds $\mathit{join}(g \otimes \mathit{id}_1) \circ [\![T]\!] \models [\![B]\!]$, that is $\mathit{join}(g \otimes [\![T]\!]) \models [\![B]\!]$ by bifunctoriality property. Since the encoding is a bijection, this is equivalent to say that for every tree T' such that

$[\![T']\!] \models [\![A]\!]$ it holds $join([\![T']\!] \otimes [\![T]\!]) \models [\![B]\!]$, that is $[\![T' \mid T]\!] \models [\![B]\!]$. By induction hypothesis, for every T' such that $T' \models_{\text{STL}} A$ it holds $T' \mid T \models_{\text{STL}} B$, that is the semantics of $T \models_{\text{STL}} A \triangleright B$.

□

Differently from STL, PGL can also describe structures with several holes and regions. In [60] we show how PGL describes contexts of tree-shaped semistructured data. In particular multi-contexts can be useful to specify properties of web-services. Consider for instance a function taking two trees and returning the tree obtained by merging their roots. Such function is represented by the term *join*, which solely satisfies the formula **join**. Similarly, the function that takes a tree and encapsulates it inside a node *labelled* by K, is represented by the term K and captured by the formula K. Moreover, the formula $\mathbf{join} \circ (\mathsf{K} \otimes (\mathbf{T} \circ \mathbf{id}_1))$ expresses all contexts of form $2 \to 1$ that place their first argument inside a K node and their second one as a sibling of such node.

4.3 Link Graph Logic (LGL).

For Λ a denumerable set of names, we consider the monoid $(\mathcal{P}_{fin}(\Lambda), \uplus, \emptyset)$, where $\mathcal{P}_{fin}(\_)$ is the finite powerset operator and $\uplus$ is the union on disjoint pairs of sets and undefined otherwise. The structures that arise from such a monoid are the link graphs. They can describe nominal resources common in many areas, such as object identifiers, location names in memory structures, channel names, and ID attributes in XML documents. But the fact that names cannot be shared implicitly it does not mean that we can refer to them or link them explicitly (e.g. object references, location pointers, fusion in fusion calculi, and IDREF in XML files). Link graphs describe connections between resources performed by means of names, i.e. *references*.

Wiring terms are a structured way to map a set of inner names X into a set of outer names Y. They are generated by the constructors: $/a : \{a\} \to \emptyset$ and ${}^{a}/_{X} : X \to a$. The closure $/a$ hides the inner name a in the outer face. The substitution ${}^{a}/_{X}$ associates all the names in the set X to the name a. We denote wirings by ω, substitutions by σ, τ, and *renamings* (i.e., bijective substitutions) by α, β. Substitution can be specialised in:

$$a \stackrel{def}{=} {}^{a}/_{\emptyset}; \qquad a \leftarrow b \stackrel{def}{=} {}^{a}/_{\{b\}}; \qquad a \leftrightarrows b \stackrel{def}{=} {}^{a}/_{\{a,b\}}.$$

The constructor a represents the introduction of a name a, term $a \leftarrow b$ the renaming of b to a, and finally $a \leftrightarrows b$ links (or fuses) a and b in the name a.

Given a signature $\mathcal{K}$ of controls K with corresponding ports $ar(\mathsf{K})$ we generate link graphs from wirings and the constructor $\mathsf{K}_{\vec{a}} : \emptyset \to \vec{a}$ with $\vec{a} = a_1, \ldots, a_k$, $\mathsf{K} \in \mathcal{K}$, and $k = ar(\mathsf{K})$. $\mathsf{K}_{\vec{a}}$ represents a resource of kind K with named ports $\vec{a}$. Any ports may be connected to other node ports via wiring compositions.

Table 4.3.1. *Additional Axioms for Link Graph Structural Congruence*

Link Axioms:	
${}^{a}/_{a} \equiv id_a$	Link Identity
$/a \circ {}^{a}/_{b} \equiv /b$	Closing renaming
$/a \circ a \equiv id_{\epsilon}$	Idle edge
${}^{b}/_{(Y \uplus a)} \circ (id_Y \otimes {}^{a}/_{X}) \equiv {}^{b}/_{Y \uplus X}$	Composing substitutions
Link Node Axiom:	
$\alpha \circ \mathsf{K}_{\vec{a}} \equiv \mathsf{K}_{\alpha(\vec{a})}$	Renaming

The structural congruence $\equiv$ for link graphs is refined in Table 1 with obvious axioms for links, modeling α-conversion and extrusion of closed names.

We assume the transparency predicate τ to be true for wiring constructors.

Given the transparency τ for each control in $\mathcal{K}$, the Link Graph Logic LGL$(\mathcal{K}, \tau)$ is $BiLog(\mathcal{P}_{fin}(\Lambda), \uplus, \emptyset, \equiv, \mathcal{K} \cup \{/a, {}^{a}/_{X}\}, \tau)$. By Theorem 3.2.5, LGL describes the link graphs precisely. The logic expresses structural spatiality for resources and strong spatiality (separation) for names, and it can therefore be viewed as a generalisation of Separation Logic for contexts and multi-ports locations. On the other side the logic can describe resources with local (hidden/private) names between resources, and in this sense the logic is a generalisation of Spatial Graph Logic [40], by considering the edges as resources.

Moreover, if we consider identity as constructor it is possible to derive:

$$a \leftarrow b \stackrel{def}{=} (a \leftrightarrows b) \circ (a \otimes id_b)$$

In LGL the formula $A \otimes B$ describes a decomposition into two *separate* link graphs (i.e., sharing no resources, names, nor connections) satisfying respectively A and B. The fact that the tensor product is only defined on link graphs with disjoint inner/outer sets of names makes of it a *spatial*, *separation* operator, in the sense that it separates the model into two distinct parts that cannot share names.

Observe that in this case, horizontal decomposition inherits the commutativity property from the monoidal tensor product. If we want a name a to be shared between separated resources, we need the sharing to be made explicit, and the sole way to do that is through the link operation. We therefore need a way to first separate the names occurring in two wirings in order to apply the tensor, and then link them back together.

As a shorthand if $W : X \to Y$ and $W' : X' \to Y'$ with $Y \subset X'$, we write $[W']W$ for $(W' \otimes id_{X' \setminus Y}) \circ W$ and if $\vec{a} = a_1, \ldots, a_n$ and $\vec{b} = b_1, \ldots, b_n$, we write $\vec{a} \leftarrow \vec{b}$ for $a_1 \leftarrow b_1 \otimes \ldots \otimes a_n \leftarrow b_n$ (and similarly for $\vec{a} \leftrightarrows \vec{b}$). It is possible to derive from the tensor product a product with sharing on $\vec{a}$. Given $G : X \to Y$ and $G' : X' \to Y'$ with $X \cap X' = \emptyset$, we choose a list $\vec{b}$ (with the same length as $\vec{a}$) of fresh names. The

composition with sharing $\vec{a}$ is:

$$G \overset{\vec{a}}{\otimes} G' \overset{def}{=} [\vec{a} \leftleftarrows \vec{b}](([\vec{b} \leftarrow \vec{a}] \circ G) \otimes G')$$

By extending this sharing to all names we can define the parallel composition $G \mid G'$ as a total operation. However, such an operator does not behave "well" with respect to the composition, as shown in [94]. In addition a direct inclusion of a corresponding connective in the logic would impact the satisfaction relation by expanding the finite horizontal decompositions to the boundless possible name-sharing decompositions. (This may be the main reason why logics describing models with name closure and parallel composition are undecidable [57].) This is due to the fact that the set of names shared by a parallel composition is not known in advance, and therefore parallel composition can only be defined using an existential quantification over the entire set of shared names.

The tensor product is well defined since all the common names $\vec{a}$ in W are renamed to fresh names, while the sharing is re-established afterwords by linking the $\vec{a}$ names with the $\vec{b}$ names.

Names can be internalised and effectively made private to a bigraph by the closure operator $/a$. The effect of composition with $/a$ is to add a new edge with no public name, and therefore to make a disappear from the outerface, and hence be completely hidden to the outside. Notice that separation is still expressed by the tensor connective, which not only separates places with an ideal line, but also makes sure that no edge – whether by visible or hidden – crosses the line. As a matter of fact, without name quantification it is not possible to build formulas that explore a link, since the latter has the effect of hiding names. For this task, we employ the name variables $x_1, ..., x_n$ and a fresh name quantification in the style of Nominal Logic [103].

$$G \models \mathsf{N} x_1, \ldots, x_n.\, A \overset{def}{=} \exists a_1 \ldots a_n \notin fn(G) \cup fn(A).\, G \models A\{x_1, \ldots\, x_n \leftarrow a_1 \ldots a_n\}$$

Using fresh name quantification we can define a notion of $\vec{a}$-linked name quantification for fresh names, whose purpose is to identify names that are linked to $\vec{a}$:

$$\vec{a}\,\mathbf{L}\,\vec{x}.\, A \overset{def}{=} \mathsf{N}\vec{x}.\, ((\vec{a} \leftleftarrows \vec{x}) \otimes \mathbf{id}) \circ A.$$

The formula above expresses that variables in $\vec{x}$ denote in A names that are linked in the term to $\vec{a}$, and the role of $(\vec{a} \leftleftarrows \vec{x})$ is to link the fresh names $\vec{x}$ with $\vec{a}$, while **id** deals with names not in $\vec{a}$. We also define a *separation-upto*, namely the decomposition in two terms that are separated apart from the link on the specific names in $\vec{a}$, which crosses the separation line.

$$A \overset{\vec{a}}{\otimes} B \overset{def}{=} \vec{a}\,\mathbf{L}\,\vec{x}.\, (((\vec{x} \leftarrow \vec{a}) \otimes \mathbf{id}) \circ A) \otimes B.$$

The idea of the formula above is that the shared names $\vec{a}$ are renamed in fresh names $\vec{x}$, so that the product can be performed and finally $\vec{x}$ is linked to $\vec{a}$ in order to actually have the sharing.

The following lemma states that the two definition are consistent.

Lemma 4.3.1 (Separation-up-to)**.** *If $g \models A \overset{\vec{x}}{\otimes} B$ with $g : \epsilon \to X$, and $\vec{x}$ is the vector of the elements in X, then there exist $g_1 : \epsilon \to X$ and $g_2 : \epsilon \to X$ such that $g \equiv g_1 \overset{\vec{x}}{\otimes} g_2$ and $g_1 \models A$ and $g_2 \models B$.*

Proof. Simply apply the definitions and observe that the identities must be necessarily id_ϵ, as the outer face of g is restricted to be X. □

The corresponding parallel composition operator is not directly definable using separation-upto, since we do not know a priori the names shared in arbitrary decompositions. However, we will show that a careful encoding is possible for the parallel composition of spatial logics with nominal resources.

4.3.1 Encoding SGL

We show that LGL can be seen as a contextual (and multi-edge) version of Spatial Graph Logic (SGL) [40]. The logic SGL expresses properties of directed edge labelled graphs G built from the empty graph nil, the edge labelled a from x to y nodes $a(x, y)$, the parallel composition of graphs $G_1 \mid G_2$, and the binding for local names of nodes $(\nu x)G$. We consider a $\mathcal{K}$ such that: there is a bijective function associating every edge label a to a distinct control $\mathsf{K}(a)$ and the arity of every control is 2 (the ports represent the starting and arrival node respectively). The resulting link graphs can be interpreted as contextual edge labelled graphs and the resulting class of ground link graphs is isomorphic to the graph model of SGL. It is a static fragment of the Ambient Logic [45] characterized by propositional connectives, spatial connectives (i.e., 0, $a[A]$, $A \mid B$), and their adjuncts (i.e., $A@a$, $A \triangleright B$). STL is quite expressive, even in the propositional case. For example, the adjunct operators express an implicit universal quantification on models, which can be used to internalise the validity problem in the model checking problem. In Table 4.3.2 we encode the graphs modeling SGL into ground link graphs and SGL formulas into LGL formulas. The encoding is parametric on a finite set X of names containing the free names of the graph under consideration. Observe that when we force the outer face of the graphs to be a fixed finite set X, the encoding of parallel composition is simply the separation-upto $\vec{a}$, where $\vec{a}$ is a list of all the elements in X. Notice also how local names are encoded into name closures (and identity).

Thanks to the Connected Normal Form provided in [94], it is easy to prove that ground link graphs featuring controls with exactly two ports are isomorphic to spatial graph models. As we impose a bijection between arrows labels and controls, the signature and the label set must have the same cardinality.

The extrusion properties of local names are guaranteed by node and link axioms.

Table 4.3.2. *Encoding Propositional SGL in LGL over two ported ground link graphs*

Spatial Graphs into Two-ported Ground Link Graphs

$[\![nil]\!]_X \stackrel{def}{=} X$

$[\![a(x,y)]\!]_X \stackrel{def}{=} \mathsf{K}(a)_{x,y} \otimes X \setminus \{x,y\}$

$[\![(\nu x)G]\!]_X \stackrel{def}{=} ((/x \otimes id_{X\setminus\{x\}}) \circ [\![G]\!]_{\{x\}\cup X})) \otimes (\{x\} \cap X)$

$[\![G \mid G']\!]_X \stackrel{def}{=} [\![G]\!]_X \stackrel{\vec{a}}{\otimes} [\![G']\!]_X$

SGL formulas into LGL formulas

$[\![\mathbf{nil}]\!]_X \stackrel{def}{=} X$

$[\![a(x,y)]\!]_X \stackrel{def}{=} \mathsf{K}(a)_{x,y} \otimes (X \setminus \{x,y\})$

$[\![\mathbf{F}]\!]_X \stackrel{def}{=} \mathbf{F}$

$[\![\phi \Rightarrow \psi]\!]_X \stackrel{def}{=} [\![\phi]\!]_X \Rightarrow [\![\psi]\!]_X$

$[\![\phi \mid \psi]\!]_X \stackrel{def}{=} [\![\phi]\!]_X \stackrel{\vec{a}}{\otimes} [\![\psi]\!]_X$

Lemma 4.3.2 (Isomorphism for spatial graphs)**.** *There exists a mapping* $(\![\ ]\!)$*, inverse to* $[\![\ ]\!]$*, such that:*

1. *For every ground link graph* g *with outer face* X *in the signature featuring a countable set of controls* K*, all with arity* 2*, it holds*

$$fn((\![g]\!)) = X \quad \text{and} \quad [\![(\![g]\!)]\!]_X \equiv g.$$

2. *For every spatial graph* G *with* $fn(G) = X$ *it holds*

$$[\![G]\!]_X : \epsilon \to X \quad \text{and} \quad (\![[\![G]\!]_X]\!) \equiv G.$$

Proof. The idea is to interpret link graphs as bigraphs without nested nodes and type $\epsilon \to \langle 1, X\rangle$. The results in [94] say that a bigraph without nested nodes and $\langle 1, X\rangle$ as outerface have the following normal form (where $Y \subseteq X$):

$$\begin{aligned} G &::= (/Z \mid id_{\langle 1,X\rangle}) \circ (X \mid M_0 \mid \ldots \mid M_{k-1}) \\ M &::= \mathsf{K}_{x,y}(a) \circ 1 \end{aligned}$$

The inverse encoding is based on such a normal form:

$$(\![(/Z \mid id_{\langle 1,X\rangle}) \circ (X \mid M_0 \mid \ldots \mid M_{k-1})]\!) \stackrel{def}{=} (\nu Z)\,(nil \mid (\![M_0]\!) \mid \ldots \mid (\![M_{k-1}]\!))$$
$$(\![\mathsf{K}_{x,y}(a) \circ 1]\!) \stackrel{def}{=} a(x,y)$$

Notice that the extrusion properties of local names correspond to node and link axioms. The encodings $[\![\ ]\!]$ and $(\![\ ]\!)$ provide a bijection, up to congruence, between graphs of SGL and ground link graphs with outer face X and built by controls of arity 2. □

The previous lemma is fundamental in proving that the soundness of the encoding for *SGL* in BiLog, stated in the following theorem.

Theorem 4.3.3 (Encoding SGL)**.** *For each graph G, finite set X containing $fn(G)$, and formula ϕ of the propositional fragment of SGL, we have that $G \models_{GL} \phi$ if and only if $[\![G]\!]_X \models ([\![\phi]\!]_X)_{\emptyset \to X}$.*

Proof. By induction on formulae of SGL. The transparency predicate is not considered here, as it is verified on every control. The basic step deals with the constants **F**, **nil** and $a(x, y)$. Case **F** follows by definition. For the case **nil**, $[\![G]\!]_X \models [\![\mathbf{nil}]\!]_X$ means $[\![G]\!]_X \models X$, that by definition is $[\![G]\!]_X \equiv X$ and so $G \equiv (\!| [\![G]\!]_X |\!) \equiv (\!| X |\!) \stackrel{def}{=} nil$, namely $G \models_{\text{SGL}} \mathbf{nil}$. For the case $a(x, y)$, to assume $[\![G]\!]_X \models [\![a(x, y)]\!]_X$ means $[\![G]\!]_X \models \mathsf{K}(a)_{x,y} \otimes X \setminus \{x, y\}$. So $G \equiv (\!| [\![G]\!]_X |\!) \equiv (\!| \mathsf{K}(a)_{x,y} \otimes X \setminus \{x, y\} |\!) \equiv a(x, y)$, that is $G \models_{\text{SGL}} a(x, y)$.

The inductive steps deal with connectives.

Case $\varphi \Rightarrow \psi$**.** To assume $[\![G]\!]_X \models [\![\varphi \Rightarrow \psi]\!]_X$ means $[\![G]\!]_X \models [\![\varphi]\!]_X \Rightarrow [\![\psi]\!]_X$; by definition this says that $[\![G]\!]_X \models [\![\varphi]\!]_X$ implies $[\![G]\!]_X \models [\![\psi]\!]_X$. By induction hypothesis, this is equivalent to say that $G \models_{\text{SGL}} \varphi$ implies $G \models_{\text{SGL}} \psi$, namely $G \models_{\text{SGL}} \varphi \Rightarrow \psi$.

Case $\varphi \mid \psi$**.** To assume $[\![G]\!]_X \models [\![\varphi \mid \psi]\!]_X$ means $[\![G]\!]_X \models [\![\varphi]\!]_X \stackrel{\vec{x}}{\otimes} [\![\psi]\!]_X$. By Lemma 4.3.1 there exists g_1, g_2 such that $[\![G]\!]_X \equiv g_1 \stackrel{\vec{x}}{\otimes} g_2$ and $g_1 \models [\![\varphi]\!]_X$ and $g_2 \models [\![\psi]\!]_X$. Let $G_1 = (\!| g_1 |\!)$ and $G_2 = (\!| g_2 |\!)$, Lemma 4.3.2 says that $[\![G_1]\!]_X \equiv g_1$ and $[\![G_2]\!]_X \equiv g_2$, and by conservation of congruence, $[\![G_1]\!]_X \models [\![\varphi]\!]_X$ and $[\![G_2]\!]_X \models [\![\psi]\!]_X$. Hence the induction hypothesis says that $G_1 \models_{\text{SGL}} \varphi$ and $G_2 \models_{\text{SGL}} \psi$. In addition $[\![G_1 \mid G_2]\!]_X \equiv [\![G_1]\!]_X \stackrel{\vec{x}}{\otimes} [\![G_2]\!]_X \equiv g_1 \stackrel{\vec{x}}{\otimes} g_2 \equiv [\![G]\!]_X$. Conclude that G admits a parallel decomposition with parts satisfying A and B, thus $G \models_{\text{SGL}} \varphi \mid \psi$.

□

In LGL it could be also possible to encode the Separation Logics on heaps: names used as identifiers of location will be forcibly separated by tensor product, while names used for pointers will be shared/linked. However we don't encode it explicitly since in the following we will encode a more general logic: the Context Tree Logic [33].

4.4 Pure bigraph Logic

We combine the structures of link graphs and place graphs to generate all *(abstract pure) bigraphs* of [80]. We take as monoid the product of link and place interfaces, i.e. $(\omega \times \mathcal{P}_{fin}(\Lambda), \otimes, \epsilon)$ where $\langle m, X \rangle \otimes \langle n, X \rangle \stackrel{def}{=} \langle m + n, X \uplus Y \rangle$ and $\epsilon \stackrel{def}{=} \langle 0, \emptyset \rangle$. We will use X for $\langle 0, X \rangle$ and n for $\langle n, \emptyset \rangle$.

As constructors for bigraphical terms we have the union of place and link graph constructors apart from the controls $\mathsf{K} : 1 \to 1$ and $\mathsf{K}_{\vec{a}} : \emptyset \to \vec{a}$, which are replaced

by the new *discrete ion* constructor, which we note $\mathsf{K}_{\vec{a}} : 1 \to \langle 1, \vec{a} \rangle$; this is a prime bigraph containing a single node with ports named $\vec{a}$ and an hole inside. Bigraphical terms thus are defined w.r.t. a control signature $\mathcal{K}$ and a set of names Λ, cf. [94] for details.

The structural congruence for bigraphs corresponds to the sound and complete axiomatisation of bigraphs of [94], its additional axioms are reported in Table 4.4.1; they are essentially a combination of the previous, there are only small differences due to the different monoid of interfaces. Namely, we define the symmetry as

$$\gamma_{I,J} \stackrel{def}{=} \gamma_{m,n} \otimes id_{X \uplus Y} \text{ where } I = \langle m, X \rangle \,,\; J = \langle n, Y \rangle$$

and we restate the node axiom taking care of the places.

Table 4.4.1. *Additional axioms for Bigraph Structural Congruence*

Symmetric Category Axioms:	
$\gamma_{I,\epsilon} \equiv id_I$	Symmetry Id
$\gamma_{I,J} \circ \gamma_{J,I} \equiv id_{I \otimes J}$	Symmetry Composition
$\gamma_{I',J'} \circ (G \otimes F) \equiv (F \otimes G) \circ \gamma_{I,J}$	Symmetry Monoid
Place Axioms:	
$join \circ (1 \otimes id_1) \equiv id_1$	Unit
$join \circ (join \otimes id_1) \equiv join \circ (id_1 \otimes join)$	Associativity
$join \circ \gamma_{1,1} \equiv join$	Commutativity
Link Axioms:	
${}^{a}/_{a} \equiv id_a$	Link Identity
$/a \circ {}^{a}/_{b} \equiv /b$	Closing renaming
$/a \circ a \equiv id_\epsilon$	Idle edge
${}^{b}/_{(Y \uplus a)} \circ (id_Y \otimes {}^{a}/_{X}) \equiv {}^{b}/_{Y \uplus X}$	Composing substitutions
Node Axiom:	
$(id_1 \otimes \alpha) \circ \mathsf{K}_{\vec{a}} \equiv \mathsf{K}_{\alpha(\vec{a})}$	Renaming

PGL excels at expressing properties of *unnamed* resources, i.e., resources accessible only by following the structure of the term. On the other hand, LGL characterises names and their links to resources, but it has no notion of locality. A combination of them ought to be useful to model nominal spatial structures, either private or public.

BiLog promises to be a good (contextual) spatial logic for (semi-structured) resources with nominal links, thanks to bigraphs' orthogonal treatment of locality and connectivity. To testify this we have proved [59] that also the recently proposed Context Logic for Trees [33] can be encoded into bigraphs. The idea of the encoding is to extend the encoding of STL with (single-hole) contexts and identified nodes.

4.4.1 Encoding CTL

In section 1.3.3 we presented the Context Tree Logic of [33]. The complete structure has also link values, but for simplicity here we restrict our attention to the fragment without them. The terms considered in CTL are constrained not to share identifiers, as the latter are locations in the memory and are used by the program to identify the node. Thus, two nodes cannot have the same identifier. This is easily obtained in bigraph terms by encoding identifiers as names and composition as tensor product (that separates them). We can encode such a structure in BiLog by lifting the application to a particular kind of composition, and similarly for the two adjuncts.

The tensor product on bigraphs is both a spatial separation (like in models of STL), and a partially-defined separation on names (like the one for pointers in separation logic). Since we deal with both names and places, we define a formula $\mathbf{id}_{\langle m,\_\rangle}$ to represent identities on places by fixing the place part of the interface and leaving the name part free.

$$\mathbf{id}_{\langle m,\_\rangle} \stackrel{def}{=} \mathbf{id}_m \otimes (\mathbf{id} \wedge \neg(\mathbf{id}_1^{\exists\otimes}))$$

Using this identity formula we can define the corresponding typed composition $\circ_{\langle m,\_\rangle}$ and typed composition adjuncts $\multimap_{\langle m,\_\rangle}$, $\multimap_{\langle m,\_\rangle}$.

We then define parallel composition with separation $*$ – both as a term constructor and as a logical connective – as follows: $D * D' \stackrel{def}{=} [join](D \otimes D')$, for D and D' prime bigraphs, and $A * B \stackrel{def}{=} (\mathbf{join} \otimes \mathbf{id}_{\langle 0,\_\rangle}) \circ (A_{\rightarrow\langle 1,\_\rangle} \otimes B_{\rightarrow\langle 1,\_\rangle})$, for A and B formulas.

This shows how BiLog for discrete bigraphs (that is bigraphs without links and closure constructors) is a generalisation of Context Tree Logic to contexts with several holes (and regions). The encodings are detailed in Table 4.4.2. We call unary bigraphs the prime bigraphs with one single hole (of type $1 \to \langle 1, Y\rangle$), that correspond to simple structured contexts. We assume as usual an bijective function from tags to controls and we use only controls with arity one (the identifier of the location).

Theorem 4.4.1 (Encoding Context Tree Logic)**.** *For each tree T and formula P of CTL It holds $T \models_T P$ if and only if $[\![T]\!] \models [\![P]\!]_P$. Also, for each context C and formula K of CTL it holds $C \models_K K$ if and only if $[\![C]\!]_C \models [\![K]\!]_K$.*

Proof. Follow the lines of Theorem 4.2.2 and 4.3.3, by structural induction on CTL formulae and by exploiting the fact that the encoding of contexts trees in unary discrete bigraphs is bijective. □

The encoding shows that the models introduced in [33] are a particular kind of discrete bigraphs with one port for each node and a number of holes and roots limited to one. Since [33] is more general than separation logic, and is used to reason about programs that manipulate tree structured memory model, we can express separation logic too and use the pure bigraph logic to reason about programs with complex query/update memory instruction, involving many locations simultaneously.

Table 4.4.2. *Encoding Context TL in BiLog over prime discrete ground bigraphs*

Trees into prime ground discrete bigraphs	Contexts into unary discrete bigraphs
$[\![\, 0 \,]\!] \stackrel{def}{=} 1$	$[\![\, - \,]\!]_C \stackrel{def}{=} id_1$
$[\![\, a_x[T] \,]\!] \stackrel{def}{=} (\mathsf{K}(a)_x \otimes fn(T)) \circ [\![\, T \,]\!]$	$[\![\, a_x[C] \,]\!]_C \stackrel{def}{=} (\mathsf{K}(a)_x \otimes fn(C)) \circ [\![\, C \,]\!]_C$
$[\![\, T_1 \mid T_2 \,]\!] \stackrel{def}{=} [\![\, T_1 \,]\!] * [\![\, T_2 \,]\!]$	$[\![\, T \mid C \,]\!]_C \stackrel{def}{=} [\![\, T \,]\!] * [\![\, C \,]\!]_C$
	$[\![\, C \mid T \,]\!]_C \stackrel{def}{=} [\![\, C \,]\!]_C * [\![\, T \,]\!]$
TL formulas into PGL formulas	**Context formulas into PGL formulas**
$[\![\, \mathit{false} \,]\!]_P \stackrel{def}{=} \mathbf{F}$	$[\![\, \mathit{false} \,]\!]_K \stackrel{def}{=} \mathbf{F}$
$[\![\, K(P) \,]\!]_P \stackrel{def}{=} [\![\, K \,]\!]_K \circ_{\langle 1,\_ \rangle} [\![\, P \,]\!]_P$	$[\![\, - \,]\!]_K \stackrel{def}{=} \mathbf{id}_1$
$[\![\, K \triangleleft P \,]\!]_P \stackrel{def}{=} [\![\, K \,]\!]_K \circ\!\!- _{\langle 1,\_ \rangle} [\![\, P \,]\!]_P$	$[\![\, P \triangleright P' \,]\!]_K \stackrel{def}{=} [\![\, P \,]\!]_P \multimap_{\langle 1,\_ \rangle} [\![\, P' \,]\!]_P$
$[\![\, P \Rightarrow P' \,]\!]_P \stackrel{def}{=} [\![\, P \,]\!]_P \Rightarrow [\![\, P' \,]\!]_P$	$[\![\, a_x[-] \,]\!]_K \stackrel{def}{=} (\mathsf{K}(a)_x) \otimes id_{\langle 0,\_ \rangle}$
	$[\![\, P \mid - \,]\!]_K \stackrel{def}{=} [\![\, P \,]\!]_P * \mathbf{id}_1$
	$[\![\, K \Rightarrow K' \,]\!]_K \stackrel{def}{=} [\![\, K \,]\!]_K \Rightarrow [\![\, K' \,]\!]_K$

4.5 Towards dynamics

The main aim of this Chapter is to show the expressive power of BiLog in describing static structures. BiLog is however able to deal with the dynamic behaviour of the model also. Essentially, this happens thanks to the contextual nature of the logic that can be used to characterise structural parametric reaction rules.

In process algebras the dynamics is often presented by *reaction* (or rewriting) rules of the form $r \longrightarrow\!\!\triangleright\, r'$, meaning that r (the *redex*) is replaced by to r' (the *reactum*) in *suitable* contexts, named *active*. A *bigraphical reactive system* is a system provided with a set of parametric reaction rules, i.e., a set S of couples (R, R'), where the bigraphs R and R' are the redex and the reactum of a parametric reaction. We say that a ground bigraph g reacts to g' (and we write $g \longrightarrow\!\!\triangleright\, g'$) if there is a couple $(R, R') \in S$, a set of names Y, an active bigraph D, and a ground bigraph d, such that $g \equiv D \circ (R \otimes id_Y) \circ d$ and $g' \equiv D \circ (R' \otimes id_Y) \circ d$.

When the model is enriched with a dynamical framework, it is natural to enrich the logic in order to catch the temporal evolution of its model. The usual way is to introduce a modality $\Diamond$ (the *next step modality*), and extend the relation $\models$ by defining '$g \models \Diamond A$ *iff* $g \longrightarrow\!\!\triangleright\, g'$ and $g' \models A$.' According to the formulation of reduction given above, we obtain

$$g \models \Diamond A \text{ iff there exist } (R, R') \in S,\ id_Y,\ D \text{ active, and } d \text{ ground; such that } g \equiv D \circ (R \otimes id_Y) \circ d,\ g' \equiv D \circ (R' \otimes id_Y) \circ d \text{ and } g' \models A. \quad (4.2)$$

In several cases, notably the bigraphical system describing CCS [96], such operators can be expressed directly by using the static BiLog. Even more interesting is the relation between activeness of controls and their transparency as it seems related

to the intensionality/extensionality of the logic. A full treatment of dynamics in BiLog, and in particular the encoding of existing logics for concurrency, is currently under investigation.

A main feature for a distributed system is mobility, or dynamics in general. In dealing with communicating and nomadic processes, the interest is not only to describe their internal structure, but also their behaviour. So far, it has been shown how BiLog is suitable to describe structures, this section is intended to study how express an system in evolution with BiLog. The usual way to express evolution with a logic is to introduce a *next step* modality ($\Diamond$), that hints how the system can evolve in the future. In general, to say that a process satisfies the formula $\Diamond A$ amounts to say that such a process can evolve into a process satisfying A. It is worth to anticipate that, depending on the details of the underlying bigraphical model, BiLog can be expressive enough to encode such a dynamical next step modality with its intentional connectives. This section shows this fact by considering a simple case, which is derived from the encoding of CCS into bigraphs introduced in [96].

We focus on the fairly small fragment of CCS considered in [31], consisting of prefix and parallel composition only. We shall let P, Q range over *processes*, and $a, \overline{a}$ range over the actions, chosen in the enumerable set *Acts* (our result work when this set is finite). The following grammar produces the syntax of the calculus.

$$\begin{array}{lcl} P & ::== & \mathbf{0} \mid \lambda.P \mid P \mid P \\ \lambda & ::== & a \mid \overline{a}. \end{array}$$

Note that the operator ν is not included, hence all the names appearing in a process are free, this fact yields the encoding to produce bigraphs with open links. The *structural congruence* is defined as the least congruence $\equiv$ on processes such that $P \mid \mathbf{0} \equiv P$, $P \mid Q \equiv Q \mid P$ and $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$. Moreover, the dynamics is given by the usual *reduction operational semantics*:

$$\frac{}{a.P \mid \overline{a}.Q \to P \mid Q} \qquad \frac{P \to Q}{P \mid R \to Q \mid R} \qquad \frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q} \tag{4.3}$$

In process calculi, dynamics is often presented by means of *reaction* (or rewriting) rules of the form $r \longrightarrow\!\!\triangleright\, r'$, meaning that r is replaced by r' in all *suitable* contexts. The terms r, r' are named *redex* and *reactum*, respectively. Here, in particular, the bigraphs we consider are built with two controls with arity 1: act and coact, for action and coaction, that produces constructors of the form act_a and coact_a, for every action a of the CCS calculus. Intuitively, cfr. [96], the reactions are expressed as

$$\mathsf{act}_a \Box_1 \mid \mathsf{coact}_a \Box_2 \longrightarrow\!\!\triangleright\, a \mid \Box_1 \mid \Box_2. \tag{4.4}$$

The rules are parametric, in the sense that the two holes ($\Box_1$ and $\Box_2$) can be filled up by any process, and the link a is introduced with the purpose of maintaining the same interface between redex and reactum. By definition redex can be replaced by the reactum in any bigraphical *active* context. The 'activeness' is defined on the

structure of contexts by a predicate δ, that is closed for ids and composition. In this particular case, such a predicate projects CCS's active contexts into bigraphs. The rules in (4.3) implies that active contexts in CCS must have the form $P \mid \Box$, hence the corresponding bigraphical context has the form $[\![P]\!] \mid \Box$, for $[\![P]\!]$ encoding of P into a bigraph. Since the encoding that is going to be introduced in this section involves ground single-rooted bigraphs with open links, the formal definition for an active context is

$$g \mid (id_1 \otimes id_Y) \tag{4.5}$$

for $g : \epsilon \to \langle 1, Z \rangle$ ground, single-rooted and with open links. Moreover Y has to be a finite set of names, viz., the outer names of the term that can fill the context. In particular, the controls act and coact are declared to be *passive*, i.e., no reaction can occur inside them.

One may wonder whether the modality $\Diamond$ is the only way to express a temporal evolution in BiLog. It turns out that, i some cases, BiLog has a built in notion of dynamics. In several cases, BiLog itself is sufficient to express the computation. One of them is the encoding of CCS, shown in the following.

As already said, we consider bigraphs built on the controls $\mathsf{act}_a, \mathsf{coact}_a$. The encoding $[\![\ ]\!]_X$ is defined with respect to a *finite* subset $X \subseteq Acts$. In particular, the encoding yields ground bigraphs with outer face $\langle 1, X \rangle$ and open links. The translation for processes is formally defined as

$$\begin{array}{rcl}
[\![\mathbf{0}]\!]_X & \stackrel{def}{=} & 1 \otimes X; \\
[\![a.P]\!]_X & \stackrel{def}{=} & (\mathsf{act}_a \overset{a}{\otimes} id_X) \circ [\![P]\!]_X; \\
[\![\bar{a}.P]\!]_X & \stackrel{def}{=} & (\mathsf{coact}_a \overset{a}{\otimes} id_X) \circ [\![P]\!]_X; \\
[\![P \mid Q]\!]_X & \stackrel{def}{=} & join \circ ([\![P]\!]_X \overset{X}{\otimes} [\![Q]\!]_X).
\end{array}$$

Where $a \in X$. With abuse of notation, the sharing/separation operator $\overset{X}{\otimes}$ stands for $\overset{\vec{a}}{\otimes}$ where $\vec{a}$ is any array of all the elements in X. Note, in particular, that the sharing tensor "$\overset{a}{\otimes} id_X$" allows the process to fill the hole in act_a (and coact_a) to perform other a actions, moreover *join* makes commutative the tensor in the encoding of parallel, in fact there is a straight correspondence between the parallel operators in the two calculi, as $[\![P \mid Q]\!]_X$ corresponds to $[\![P]\!]_X \mid [\![Q]\!]_X$. A first result about the encoding is that it is bijective on prime ground bigraphs with open links, as stated in the following lemma.

Lemma 4.5.1 (Adding Names)**.** *If P has got x in its outer names, then $P \mid x \equiv P$.*

Proof. Express the parallel in terms of renamings, linkings and tensor product, obtaining, and use the axioms of [94]. Assume that $P : \langle m, X \rangle \to \langle n, \{x\} \cup Y \rangle$, and

$y \notin \{x\} \cup Y$; then

$$\begin{array}{lcll} P \mid x & \equiv & (id_{\langle n,Y\rangle} \otimes (x \leftleftarrows y)) \circ (P \otimes ((y \leftarrow x) \circ x)) & \\ & \equiv & (id_{\langle n,Y\rangle} \otimes (x \leftleftarrows y)) \circ (P \otimes y) & \text{by 3rd link axiom} \\ & \equiv & (id_{\langle n,Y\rangle} \otimes (x \leftleftarrows y)) \circ (id_{\langle n,Y\rangle} \otimes id_x \otimes y) \circ (P \otimes id_\epsilon) & \text{by bifunctoriality} \\ & \equiv & ((id_{\langle n,Y\rangle} \circ id_{\langle n,Y\rangle}) \otimes ((x \leftleftarrows y) \circ (id_x \otimes y))) \circ P & \text{by bifunctoriality} \\ & \equiv & (id_{\langle n,Y\rangle} \otimes id_x) \circ P & \text{by 2nd link axiom} \\ & \equiv & P & \end{array}$$

□

Lemma 4.5.2 (Bijective Translation)**.** *For every finite subset $X \subseteq Acts$, then*

1. *The translation $[\![\cdot]\!]_X$ is surjective on prime ground bigraphs with outerface $\langle 1, X\rangle$ and open links.*

2. *For every couple of processes P, Q and for every finite subset $X \subseteq Acts$ including the free names of P, Q it holds: $P \equiv Q$ if and only if $[\![P]\!]_X \equiv [\![Q]\!]_X$.*

Proof. We prove point (1) by showing that every prime ground bigraph with outerface $\langle 1, X\rangle$ has at least one pre-image for the translation $[\![\cdot]\!]_X$. We proceed by induction on the number of nodes in the bigraphs. First we recall the connected normal form for bigraphs. In [94] is proved a Theorem stating that every prime ground bigraph G with outerface $\langle 1, X\rangle$ and open links has the following Connected Normal Form:

$$\begin{array}{lll} G & ::= X \mid F & \\ F & ::= M_1 \mid \ldots \mid M_k & \\ M & ::= (\mathsf{K}_a \mid id_Y) \circ F & (\text{for } \mathsf{K}_a \in \{\mathsf{act}_a, \mathsf{coact}_a\}) \end{array}$$

The base of induction is the bigraph X, and clearly $[\![\mathbf{0}]\!]_X = X$. For the inductive step, consider a bigraph G with at least one node. This means $G = X \mid ((\mathsf{K}_a \mid id_Y) \circ F) \mid G'$. Without losing generality, we assume $\mathsf{K}_a = \mathsf{act}_a$, and, by Proposition 4.5.1, we obtain

$G = (\mathsf{act}_a \mid id_X) \circ (X \mid F) \mid (X \mid G')$. Now, the induction says that there exist P and Q such that $[\![P]\!]_X = X \mid F$ and $[\![Q]\!]_X = X \mid G'$, hence we conclude $[\![a.P \mid Q]\!]_X = G$.

The forward implication of point (2) is proved by showing that the translation is sound with respect to the rules of congruence in CCS. This has been already proved in [94], where the parallel $\mid$ between bigraphs is shown to be commutative and associative, and to have 1 as a unit. Moreover, by Proposition 4.5.1, the bigraph $1 \otimes X$ is the unit for the parallel operator on prime ground bigraphs with outerface $\langle 1, X\rangle$.

The following claim, stated in [96], is the crucial step in proving the reverse implication of point (2). Its proof considers the discrete normal for bigraphs.

Claim 4.5.3. *If G_i $(i = 1 \dots m)$ and F_j $(j = 1 \dots n)$ are ground molecules and $G_1 \mid \dots \mid G_m \equiv F_1 \mid \dots \mid F_n$, then $m = n$ and $G_i \equiv F_{\pi(i)}$ for some permutation π on m.*

The proof of the reverse implication of point (2) proceeds by induction on the structure of P. The base of induction is $P = \mathbf{0}$, in this case the statement is verified as to assume $[\![Q]\!]_X \equiv [\![\mathbf{0}]\!]_X = X$ implies $Q \equiv \mathbf{0} \mid \dots \mid \mathbf{0}$. For the inductive step let $P \equiv a_1.P_1 \mid \dots \mid a_m.P_m$ for any $m \geq 1$, and assume $[\![Q]\!] \equiv [\![P]\!]$. Furthermore we have $Q \equiv b_1.Q_1 \mid \dots \mid b_n.Q_n$, then

$$\begin{aligned} [\![P]\!]_X = \ & (\mathsf{act}_{a_1} \overset{a_1}{\otimes} id_X) \circ [\![P_1]\!]_X \mid \dots \mid (\mathsf{act}_{a_m} \overset{a_m}{\otimes} id_X) \circ [\![P_m]\!]_X \\ [\![Q]\!]_X = \ & (\mathsf{act}_{b_1} \overset{b_1}{\otimes} id_X) \circ [\![Q_1]\!]_X \mid \dots \mid (\mathsf{act}_{b_m} \overset{b_m}{\otimes} id_X) \circ [\![Q_m]\!]_X \end{aligned}$$

Since the two translations are both a parallel compositions of ground molecules, the previous claim says that $m = n$, and there exists a permutation π on m such that $a_i \equiv a_{\pi(i)}$ and $[\![Q_i]\!] \equiv [\![P_{\pi(i)}]\!]$. By induction $Q_i \equiv P_{\pi(i)}$, hence $Q \equiv P$. □

In [96] it is proved that the translation preserves and reflects the reactions, that is: $P \longrightarrow\!\triangleright P'$ *if and only if* $[\![P]\!] \longrightarrow\!\triangleright [\![P']\!]$.

The reaction rules are defined as

$$(\mathsf{act}_a \mid id_{Y_1}) \mid (\mathsf{coact}_a \mid id_{Y_2}) \longrightarrow\!\triangleright a \mid id_{\langle 1, Y_1 \rangle} \mid id_{\langle 1, Y_2 \rangle}. \tag{4.6}$$

It can be mildly sugared to obtain the rule introduced in (4.4)

Moreover, the active contexts introduced in 4.5 can be rephrased as

$$g \mid \square$$

where g is a single-rooted ground bigraph and with open links. It is easy to conclude that the most general context ready to react has the form

$$\square_0 \mid \mathsf{act}_a \square_1 \mid \mathsf{coact}_a \square_2 \mid\!\longrightarrow\!\triangleright \square_0 \mid \square_1 \mid \square_2$$

and the hole $\square_0$ is filled in with single-rooted ground bigraphs with open links, whereas the holes $\square_1$ and $\square_2$ can be filled in with ground bigraphs. Note that such a reduction is compositional as respect to the parallel operator. In case of the CCS translation, the a reacting bigraphs can be further characterized, as it is shown in the next lemma. In particular, the lemma shows that every reacting $[\![P]\!]_X$ can be decomposed into a redex and a bigraph with a well defined structure, which is composed with a reactum in order to obtain the result of the reaction. The Redex and the Reactum are formally outlined in Table 4.5.1, and they will be the key point to express the next step with BiLog. Note that y_1 and y_2 of the definition in Table 4.5.1 have to be disjoint with X, Y_1 and Y_2. They are useful for join the action with the corresponding coaction.

Table 4.5.1. *Reacting Contexts for CCS*

Bigraphs:

$$Redex_a^{y_1,y_2,Y_1,Y_2} \stackrel{def}{=} W \circ (id_Y \otimes join) \circ (id_Y \otimes join \otimes id_1) \circ \{((y_1 \leftarrow a) \otimes id_1) \circ \circ \mathsf{act}_a \otimes id_{Y_1} \otimes ((y_2 \leftarrow a) \otimes id_1) \circ \mathsf{coact}_a \otimes id_{Y_2} \otimes id_{\langle 1,X\rangle}\}$$

$$React_a^{Y_1,Y_2} \stackrel{def}{=} W' \circ (id_{Y'} \otimes join) \circ (id_{Y'} \otimes join \otimes id_1)$$

Wirings:

$$W \stackrel{def}{=} ((X \leftleftarrows Y_1) \otimes id_1) \circ (id_{Y_1} \otimes (X \leftleftarrows Y_2) \otimes id_1) \circ (id_{Y_1} \otimes id_{Y_2} \otimes id_{X\setminus\{a\}} \otimes \otimes (a \leftleftarrows y_1) \otimes id_1) \circ (id_{Y_1} \otimes id_{Y_2} \otimes id_{X\setminus\{a\}} \otimes id_{\{y_1\}} \otimes (a \leftleftarrows y_2) \otimes id_1)$$

$$W' \stackrel{def}{=} ((X \leftleftarrows Y_1) \otimes id_1) \circ (id_{Y_1} \otimes (X \leftleftarrows Y_2) \otimes id_1)$$

Supporting Sets:

$$Y \stackrel{def}{=} \{y_1, y_2\} \cup Y_1 \cup Y_2 \cup X$$

$$Y' \stackrel{def}{=} Y_1 \cup Y_2 \cup X$$

Lemma 4.5.4 (Reducibility)**.** *Given a CCS process P, the following are equivalent.*

1. *The translation $[\![P]\!]_X$ can perform the reduction $[\![P]\!]_X \longrightarrow\!\triangleright G$.*
2. *There exist three bigraphs $G_1, G_2, G_3 : \epsilon \to \langle 1, X\rangle$ and $a \in X$, such that*

$$[\![P]\!]_X \equiv ((\mathsf{act}_a \mid id_X) \circ G_1) \mid ((\mathsf{coact}_a \mid id_X) \circ G_2) \mid G_3$$

   and $G \equiv G_1 \mid G_2 \mid G_3$.
3. *There exist the actions $a \in X$ and $y_1, y_2 \notin X$, and two mutually disjoint subsets $Y_1, Y_2 \subseteq Acts$ with the same cardinality as X, but disjoint with X, y_1 and y_2, and there exist the bigraphs $H_i : \epsilon \to \langle 1, Y_i\rangle$, for $i = 1, 2$, and $H_3 : \epsilon \to \langle 1, X\rangle$ with open links, such that*

$$[\![P]\!]_X \equiv Redex_a^{y_1,y_2,Y_1,Y_2} \circ (H_1 \otimes H_2 \otimes H_3)$$

   and

$$G \equiv React_a^{Y_1,Y_2} \circ (H_1 \otimes H_2 \otimes H_3),$$

   where $Redex_a^{y_1,y_2,Y_1,Y_2}$, $React_a^{Y_1,Y_2}$ are defined in Table 4.5.1.

Proof. First we prove that points (1) and (2) are equivalent. Assume that the bigraph $[\![P]\!]_X$ can perform a reaction. This means that $[\![P]\!]_X \equiv ((\mathsf{act}_a \mid id_{Y_1}) \circ G'_1) \mid ((\mathsf{coact}_a \mid id_{Y_2}) \circ G'_2) \mid G'_3$ and that $G \equiv a \mid G'_1 \mid G'_2 \mid G'_3$ for some suitable ground bigraphs G'_1, G'_2 and G'_3 and an action $a \in X$. Since the type of both $[\![P]\!]_X$ and G is $\epsilon \to \langle 1, X\rangle$, we have $G \equiv (X \mid G'_1) \mid (X \mid G'_2) \mid (X \mid G'_3)$ and $[\![P]\!]_X \equiv ((\mathsf{act}_a \mid id_X) \circ (X \mid G'_1)) \mid ((\mathsf{coact}_a \mid id_X) \circ (X \mid G'_2)) \mid (X \mid G'_3)$ by applying Proposition 4.5.1. Then we define G_i to be $X \mid G'_i$ for $i = 1, 2, 3$, and we conclude $G \equiv G_1 \mid G_2 \mid G_3$ and $[\![P]\!]_X \equiv ((\mathsf{act}_a \mid id_X) \circ G_1) \mid ((\mathsf{coact}_a \mid id_X) \circ G_2) \mid G_3$.

We prove now that point (2) implies point (3). Assume that $[\![P]\!]_X \equiv ((\mathsf{act}_a \mid id_X) \circ G_1) \mid ((\mathsf{coact}_a \mid id_X) \circ G_2) \mid G_3$ and $G \equiv G_1 \mid G_2 \mid G_3$, with $G_1, G_2, G_3 : \epsilon \to \langle 1, X \rangle$. According to the definition of the parallel operator, we chose two actions $y_1, y_2 \notin X$ and the mutually disjoint subsets $Y_1, Y_2 \subseteq Acts$ that have the same cardinality as X, but are disjoint with X, y_1, y_2, and we have

$$[\![P]\!]_X \equiv W \circ (id_Y \otimes join) \circ (id_Y \otimes join \otimes id_1) \circ \{((y_1 \leftarrow a) \otimes \\ \otimes id_{\langle 1,Y_1 \rangle}) \circ (\mathsf{act}_a \otimes id_{Y_1}) \circ ((Y_1 \leftarrow X) \otimes id_{\langle 1,Y_2 \rangle}) \circ G_1 \otimes ((y_2 \leftarrow a) \otimes \\ \otimes id_1) \circ (\mathsf{coact}_a \otimes id_{Y_2}) \circ ((Y_2 \leftarrow X) \otimes id_1) \circ G_2 \otimes G_3\}$$

and

$$G \equiv W' \circ (id_{Y'} \otimes join) \circ (id_{Y'} \otimes join \otimes id_1) \circ \\ \circ \{((Y_1 \leftarrow X) \otimes id_{\langle 1,Y_2 \rangle}) \circ G_1 \otimes ((Y_2 \leftarrow X) \otimes id_1) \circ G_2 \otimes G_3\}$$

where $Y = \{y_1\} \cup Y_1 \cup \{y_2\} \cup Y_2 \cup X$ and $Y' = Y_1 \cup Y_2 \cup X$. The bigraphs W and W' are defined in Table 4.5.1, they both link the subsets Y_1 and Y_2 with X, moreover W links y_1 and y_2 with a. Thanks to the bifunctoriality property, we rewrite $[\![P]\!]_X$ as

$$W \circ (id_Y \otimes join) \circ (id_Y \otimes join \otimes id_1) \circ \{((y_1 \leftarrow a) \otimes id_1) \circ \\ \circ \mathsf{act}_a \otimes id_{Y_1} \otimes ((y_2 \leftarrow a) \otimes id_1) \circ \mathsf{coact}_a \otimes id_{Y_2} \otimes G_3 \} \circ \\ \circ \{ ((Y_1 \leftarrow X) \otimes id_1) \circ G_1 \otimes ((Y_2 \leftarrow X) \otimes id_1) \circ G_2 \},$$

and, again by bifunctoriality property, we rewrite it as

$$W \circ (id_Y \otimes join) \circ (id_Y \otimes join \otimes id_1) \circ \{((y_1 \leftarrow a) \otimes id_1) \circ \\ \circ \mathsf{act}_a \otimes id_{Y_1} \otimes ((y_2 \leftarrow a) \otimes id_1) \circ \mathsf{coact}_a \otimes id_{Y_2} \otimes id_{\langle 1,X \rangle} \} \circ \\ \circ \{ ((Y_1 \leftarrow X) \otimes id_1) \circ G_1 \otimes ((Y_2 \leftarrow X) \otimes id_1) \circ G_2 \otimes G_3 \}.$$

We conclude point (3) by defining $H'_i = ((Y_i \leftarrow X) \otimes id_1) \circ G_i$ for $i = 1, 2$, and $H_3 = G_3$. Note that the three bigraphs G_i and H_i have open links as so does $[\![P]\!]_X$. Finally, we prove that point (3) implies the point (2) by reversing the previous reasoning. □

By following the ideas of [96] it is easy to demonstrate that there is an exact match between reaction relations generated in CCS and in the bigraphical system, in the sense of the following lemma.

Proposition 4.5.5 (Matching Reactions)**.** *If X is a finite set of names, then*

$$P \to Q \quad \text{if and only if} \quad [\![P]\!]_X \multimap [\![Q]\!]_X$$

for every CCS process P and Q such that $Act(P), Act(Q) \subseteq X$.

Proof. For the forward direction, we proceed by induction on the number of the rules applied in the derivation for $P \to Q$ in CCS. The base of the induction is the only rule without premixes, that means P is $a.P_1 \mid \overline{a}.P_2$ and Q is $P_1 \mid P_2$. The translation is sound as regards this rule, since the reactive system says

$$((\mathsf{act}_a \mid id_X) \circ [\![P_1]\!]_X) \mid ((\mathsf{coact}_a \mid id_X) \circ [\![P_2]\!]_X) \longrightarrow\!\!\triangleright\, X \mid [\![P_1]\!]_X \mid [\![P_2]\!]_X.$$

The induction step considers two cases. First, assume that $P \to Q$ is derived from $P' \to Q'$, where P is $P' \mid R$ and Q is $Q' \mid R$. Then the induction says that $[\![P']\!]_X \longrightarrow\!\!\triangleright\, [\![Q']\!]_X$, hence $[\![P']\!]_X \mid [\![R]\!]_X \longrightarrow\!\!\triangleright\, [\![Q']\!]_X \mid [\![R]\!]_X$. We conclude $[\![P]\!]_X \longrightarrow\!\!\triangleright\, [\![Q]\!]_X$, as $[\![P]\!]_X$ is $[\![P']\!]_X \mid [\![R]\!]_X$ and $[\![Q]\!]_X$ is $[\![Q']\!]_X \mid [\![R]\!]_X$. Second, assume that $P \to Q$ is derived from the congruences $P \equiv P'$ and $Q' \equiv Q$, and from the transition $P' \to Q'$. We deduce $[\![P]\!]_X \equiv [\![P']\!]_X$ and $[\![Q']\!]_X \equiv [\![Q]\!]_X$ by Lemma 4.5.2, and $[\![P']\!]_X \longrightarrow\!\!\triangleright\, [\![Q']\!]_X$ by induction hypothesis. We conclude $[\![P]\!]_X \longrightarrow\!\!\triangleright\, [\![Q]\!]_X$, since the reduction is defined up to congruence.

For the reverse implication, we assume $[\![P]\!]_X \longrightarrow\!\!\triangleright\, [\![Q]\!]_X$. Then Lemma 4.5.4 says that there exist the bigraphs $G_1, G_2, G_3 : \epsilon \to \langle 1, X \rangle$ and the name $a \in X$ such that $[\![P]\!]_X \equiv ((\mathsf{act}_a \mid id_X) \circ G_1) \mid ((\mathsf{coact}_a \mid id_X) \circ G_1) \mid G_3$ and $G \equiv G_1 \otimes G_2 \otimes G_3$. Now, Lemma 4.5.2 says that for every $i = 1, 2, 3$ there exists a CCS process P_i such that $[\![P_i]\!]$ corresponds to G_i, hence $[\![P]\!] \equiv [\![a.P_1 \mid \overline{a}.P_2 \mid P_3]\!]$ and $[\![Q]\!] \equiv [\![P_1 \mid P_2 \mid P_3]\!]$. Again, Lemma 4.5.2 says that $P \equiv a.P_1 \mid \overline{a}.P_2 \mid P_3$ and $Q \equiv P_1 \mid P_2 \mid P_3$, then we conclude $R \to Q$. □

It can be proved an even stronger result, that is if a CCS translation reacts to a bigraph, then such a bigraph is a CCS translation as well. The fact is formalized in the lemma below.

Proposition 4.5.6 (Conservative Reaction)**.** *For every CCS process P such that $[\![P]\!]_X \longrightarrow\!\!\triangleright\, G$, there exists a CCS process Q such that $[\![Q]\!]_X = G$ and $P \to Q$.*

Proof. Assume that $[\![P]\!]_X \longrightarrow\!\!\triangleright\, G$, then the point (2) of Lemma 4.5.4 says that G has type $\epsilon \to \langle 1, X \rangle$ and open links, since so does $[\![P]\!]_X$. This means, by Lemma 4.5.2, that there exists a process Q such that $[\![Q]\!]_X \equiv G$. We conclude $P \to Q$ by Lemma 4.5.5. □

The logic $\mathcal{L}_{spat}$ can be encoded in a suitable instantiation of BiLog, without using the modality defined in (4.2). It is sufficient to instantiate the logic BiLog$(M, \otimes, \epsilon, \Theta, \equiv, \tau)$ in order to obtain the bigraphical encoding of CCS. We define Θ to be composed by the standard constructor for a bigraphical system with $\mathcal{K} = \{\mathsf{act}, \mathsf{coact}\}$, and predicate τ to be always true. The fact that the predicate of transparency is verified on every term is determinant for the soundness of the logic encoding we are describing.

We rephrase informally what stated in Lemma 4.5.4. The set of reactions in CCS are determined by couples of the form $(Redex_a, Reactum_a)$ for every $a \in X$,

and every reacting process is characterized by

$[\![P]\!]_X \longrightarrow\!\!\triangleright\, [\![Q]\!]_X$ iff there exists a bigraph g and $a \in X$ such that
$[\![P]\!]_X \equiv \mathit{Redex}_a \circ g$ and $[\![Q]\!]_X \equiv \mathit{Reactum}_a \circ g$.

Since in this case τ is always true, BiLog logic can fully describe the structure of a term. In particular, it is possible to define a characteristic formula for every redex and reactum, simply by rewriting every bigraphical constructor and operator with the correspondent logical constant in their bigraphical encodings. For the new names y_1, y_2, and the new subsets Y_1, Y_2, we denote with $\mathbf{Redex}_a^{y_1,y_2,Y_1,Y_2}$ and $\mathbf{React}_a^{Y_1,Y_2}$ the characteristic formulas of $\mathit{Redex}_a^{y_1,y_2,Y_1,Y_2}$ and $\mathit{React}_a^{Y_1,Y_2}$ respectively. Clearly, $G \models \mathbf{Redex}_a^{y_1,y_2,Y_1,Y_2}$ if and only if $G \equiv \mathit{Redex}_a^{y_1,y_2,Y_1,Y_2}$, and the same for the reactum. This has a prominent role in defining the encoding of the temporal modality in BiLog.

Table 4.5.2. *Encoding of $\mathcal{L}_{spat}$ into BiLog*

Encodings:
$[\![0]\!]_X \stackrel{def}{=} X \otimes \mathbf{1}$
$[\![\neg A]\!]_X \stackrel{def}{=} \neg [\![A]\!]_X$
$[\![A \wedge B]\!]_X \stackrel{def}{=} [\![A]\!]_X \wedge [\![B]\!]_X$
$[\![A \mid B]\!]_X \stackrel{def}{=} \mathbf{join} \circ ([\![A]\!]_X \overset{X}{\otimes} [\![B]\!]_X)$
$[\![A \triangleright B]\!]_X \stackrel{def}{=} И Y.\, (((Y \leftarrow X) \otimes \mathbf{id}_1) \circ \mathbf{A}_X) \multimap (\mathbf{join} \circ ((X \leftleftarrows Y) \otimes \mathbf{id}_1) \circ\!\!- [\![B]\!]_X)$
$[\![\Diamond A]\!]_X \stackrel{def}{=} \bigvee_{a \in X} И y_1.y_2.Y_1.Y_2.\, \mathbf{Redex}_a^{y_1,y_2,Y_1,Y_2} \circ [(\mathbf{React}_a^{Y_1,Y_2} \circ\!\!- [\![A]\!]_X) \wedge \mathbf{Triple}]$
Supporting Formulae:
$\mathbf{Open} \stackrel{def}{=} \neg И x.\, \Diamond(/x \circ \mathbf{T})$
$\mathbf{A}_X \stackrel{def}{=} [\![A]\!]_X \wedge \mathbf{T}_{\epsilon \to \langle 1, Y_2 \rangle} \wedge \mathbf{Open}$
$\mathbf{Triple} \stackrel{def}{=} \mathbf{T}_{\epsilon \to \langle 1, Y_1 \rangle} \otimes \mathbf{T}_{\epsilon \to \langle 1, Y_2 \rangle} \otimes \mathbf{T}_{\epsilon \to \langle 1, X \rangle}$

Formally, the encoding is defined as described in Table 4.5.2. The encodings for the logical connectives and the spatial composition are self-explanatory, in particular note that the spatial composition requires the sharing of the names in X. It correspond to a logical parallel operator, in the case that the set of names of bigraphs is fixed and finite. In the encoding for $\triangleright$ we introduce an auxiliary notation. Intuitively, the formula $\mathbf{A}_X$ is defined to constrain a bigraph to come from an encoding of a CCS process and to satisfy $[\![A]\!]_X$. In fact, $G \models \mathbf{A}_X$ means that G satisfies $[\![A]\!]_X$, moreover it has type $\epsilon \to \langle 1, X \rangle$ and finally its links are open, since a bigraph satisfies **Open** only if no closure appears in any of its decompositions, note the power of the somewhere operator. We will show that a bigraph satisfies $[\![P]\!] \models [\![A \triangleright B]\!]$ if it satisfies $[\![B]\!]_X$ if it is connected in parallel with any encoding of a CCS process satisfying A.

On the other side, in the encoding for the temporal modality $\Diamond$ the supporting formula **Triple** is satisfied by processes that are the composition of three single-rooted ground bigraphs whose outerfaces have the same number of names as X. We

will show that to say that a process satisfies $[\![\Diamond A]\!]_X$ amounts to say that it is the combination of a particular redex with a bigraph that satisfies the requirement of Lemma 4.5.4, and moreover that the corresponding reactum satisfies $[\![A]\!]_X$.

The main result of the section is formalized in the lemma below. It expresses the semantical equivalence between $\mathcal{L}_{spat}$ and its encoding in BiLog. Note in particular the requirement for a finite set of actions performable by the CCS processes. Such a limitation is not due to the presence of the next step operator, indeed looking carefully at the proof, one can see that the induction step for the temporal operator still holds in the case of a not-finite set of actions. On the contrary, the limitation is due to the adjoint operator $\triangleright$. In fact we need to bound the number of names that is shared between the processes. This happens because of the different choice for the logical product operator in BiLog. On one hand, the spatial logic had the parallel operator built in. This means that the logic do not care about the names that are actually shared between the processes. On the other hand, BiLog has a strong control on the names shared between two processes, and one needs to know them with accuracy.

Proposition 4.5.7. *If set of actions Acts is bounded to be a finite set X, then*

$$P \models_{spat} A \quad \text{if and only if} \quad [\![P]\!]_X \models [\![A]\!]_X.$$

for every process P with actions in X.

Proof. The proposition is proved by induction on the structure of formulas. The base of induction is the formula 0. To assume that $[\![P]\!]_X \models [\![0]\!]_X$ means $[\![P]\!]_X \equiv X \otimes 1$, that correspond to $P \equiv \mathbf{0}$, namely $P \models_{Spat} 0$. Then the inductive step deals with the connectives. The treatments of $\neg$, $\wedge$ and $|$ are similar, so we focus on the case of the parallel operator.

Case $A \mid B$. To say $[\![P]\!]_X \models [\![A \mid B]\!]_X$ means that there exist two bigraphs g_1, g_2, with $g_1 \models [\![A]\!]_X$ and $g_1 \models [\![B]\!]_X$, such that

$$[\![P]\!]_X \equiv \mathit{join} \circ (g_1 \overset{X}{\otimes} g_2)$$

Note that g_1, g_2 must have type $\epsilon \to \langle 1, X \rangle$ and open links, as so does $[\![P]\!]_X$. By Lemma 4.5.2, there exist two processes Q_1 and Q_2 such that $[\![Q_1]\!]$ and $[\![Q_2]\!]$ are g_1 and g_2, respectively. Then conclude

$$[\![P]\!]_X \equiv \mathit{join} \circ ([\![Q_1]\!]_X \overset{X}{\otimes} [\![Q_2]\!]_X)$$

that means $P \equiv Q_1 \mid Q_2$, again by Lemma 4.5.2. Moreover, the induction hypothesis says that $Q_1 \models A$ and $Q_2 \models B$, hence $P \models_{spat} A \mid B$.

Case $A \triangleright B$. Assume $[\![P]\!]_X \models [\![A \triangleright B]\!]_X$, then by definition there exists a fresh set Y such that for every G satisfying $(((Y \leftarrow X) \otimes \mathbf{id}_1) \circ \mathbf{A}_X)$ it holds

$$[\![P]\!]_X \otimes G \models \mathbf{join} \circ ((X \leftleftarrows Y) \otimes \mathbf{id}_1) \circ\!\!- [\![B]\!]_X$$

that is

$$\mathbf{join} \circ ((X \leftleftarrows Y) \otimes id_1) \circ (\llbracket P \rrbracket_X \otimes G) \models \llbracket B \rrbracket_X \tag{4.7}$$

Now $G \models (((Y \leftarrow X) \otimes \mathbf{id}_1) \circ \mathbf{A}_X)$ means that there is $g \models \mathbf{A}_X$ such that $G \equiv ((Y \leftarrow X) \otimes \mathbf{id}_1) \circ g$. As previously discussed (cfr. the introduction to the current proposition) $g \models \mathbf{A}_X$ says that $g \models \llbracket A \rrbracket_X$ and that g is a bigraph with open link and type $\epsilon \to \langle 1, X \rangle$. By Lemma 4.5.2, g is $\llbracket Q \rrbracket_X$ for some CCS process Q whose actions are in X.

Hence, as the set of actions *Acts* corresponds to X, we can rephrase (4.7) by saying that for *every* CCS process Q such that $\llbracket Q \rrbracket_X \models \llbracket A \rrbracket_X$ it holds

$$\mathbf{join} \circ ((X \leftleftarrows Y) \otimes id_1) \circ (\llbracket P \rrbracket_X \otimes ((Y \leftarrow X) \otimes \mathbf{id}_1) \circ \llbracket Q \rrbracket_X) \models \llbracket B \rrbracket_X$$

that is $\llbracket P \mid Q \rrbracket_X \models \llbracket B \rrbracket_X$. Then, the induction hypothesis says that for every Q, if $Q \models_{spat} A$ then $P \mid Q \models_{spat} B$, namely $P \models_{spat}$.

Case $\Diamond A$. to assume $\llbracket P \rrbracket_X \models \llbracket \Diamond A \rrbracket_X$ signifies that there exists an action $a \in X$ such that

$$\llbracket P \rrbracket_X \equiv Redex^{y_1, y_2, Y_1, Y_2} \circ H \tag{4.8}$$

where y_1, y_2 are fresh names, Y_1, Y_2 are fresh subsets with the same cardinality as X, and H is a bigraph satisfying

$$H \models (\mathbf{React}_a^{Y_1, Y_2} \circ\!\!- \llbracket A \rrbracket_X) \wedge \mathbf{Triple}. \tag{4.9}$$

In particular, Property (4.9) amounts to assert the two following points.

1. It holds $H \models \mathbf{React}_a^{Y_1, Y_2} \circ\!\!- \llbracket A \rrbracket_X$, that is

$$React_a^{Y_1, Y_2} \circ H \models \llbracket A \rrbracket_X. \tag{4.10}$$

2. It holds $H \models \mathbf{T}_{\epsilon \to \langle 1, Y_1 \rangle} \otimes \mathbf{T}_{\epsilon \to \langle 1, Y_2 \rangle} \otimes \mathbf{T}_{\epsilon \to \langle 1, X \rangle}$, that is

$$H \equiv H_1 \otimes H_2 \otimes H_3 \tag{4.11}$$

with $H_i : \epsilon \to \langle 1, Y_i \rangle$, for $i = 1, 2$, and $H_3 : \epsilon \to \langle 1, X \rangle$.

Now, by (4.8) and (4.11), we have $\llbracket P \rrbracket_X \equiv Redex^{y_1, y_2, Y_1, Y_2} \circ (H_1 \otimes H_2 \otimes H_3)$, that means $\llbracket P \rrbracket_X \longrightarrow\!\!\triangleright React_a^{Y_1, Y_2} \circ (H_1 \otimes H_2 \otimes H_3)$ by Lemma 4.5.4. Furthermore, the bigraphs H_1, H_2, H_3 have open links, as so does $\llbracket P \rrbracket_X$. Hence Lemma 4.5.2 says that there exists the CCS process Q such that $\llbracket Q \rrbracket_X$ corresponds to $React_a^{Y_1, Y_2} \circ (H_1 \otimes H_2 \otimes H_3)$, hence $P \to Q$ by Proposition 4.5.5. Finally, (4.10) says that $\llbracket Q \rrbracket_X \models \llbracket A \rrbracket_X$, and this means $Q \models_{spat} A$ by induction hypothesis. We conclude that $\llbracket P \rrbracket_X \models \llbracket \Diamond A \rrbracket_X$ is equivalent to $P \to Q$ with $Q \models_{spat} A$, namely $P \models_{spat} \Diamond A$. □

Part III

Decidability with Quantifiers and Name Abstraction

Chapter 5

Spatial Logics for Abstract Trees

In this Chapter we present a model of abstract trees, that is unordered trees with restricted names. This model is used in [38] and it is an extension of the unordered labelled tree model. Then we introduce the (still static) Spatial Logic describing such models and we introduce some notion of quantifiers on names. In the following chapters we will study decidability of various fragments of this logic. We choose a minimal fragment of the Ambient Logic, but the techniques we present should apply to every logic which uses Cardelli and Gordon revelation and hiding operators, and Gabbay and Pitts freshness quantifier. We start from the static fragment of ambient logic that Calcagno, Cardelli and Gordon proved to be decidable. We prove that the addition of a hiding quantifier makes the logic undecidable. Hiding can be decomposed as freshness plus revelation. Quite surprisingly, freshness alone is decidable, but revelation alone is not.

The term *Spatial Logics* (SL) has been recently used to refer to logics equipped with the composition-separation operator $A \mid B$. Spatial logics are emerging as an interesting tool to describe properties of several structures. Models for spatial logics include computational structures such as heaps [106, 101], trees [37], trees with hidden names [38], graphs [40], concurrent objects [30], as well as process calculi such as the π-calculus [28, 29] and the Ambient Calculus [43, 45].

In all these structures, a notion of *name restriction* arises. The restriction $(\nu n)\, P$ (in π-calculus notation) of a name n in a structure P is a powerful abstraction mechanism that can be used to model information that is protected by the computational model, such as hidden encryption keys [3], the actual variable names in λ-calculus, object identifiers in object calculi, and locations in a heap. Here "protected" means that no public name can ever clash with one that is protected, and that any observable behavior may depend on the equality between two names, but not on the actual value of a protected name.

Reasoning about protected names is difficult because they are "anonymous". Cardelli and Gordon suggest an elegant solution to this problem [44]. They adopt Gabbay and Pitts fresh name quantification, originally used for binder manipulation and Nominal Logics [103, 70], and combine it with a new operator, *revelation*, which

allows a public name to be used to denote a protected one. The combination of freshness quantification and revelation gives rise to a new quantifier, *hidden name quantification*, which can be used to describe properties of restricted names in a natural way.

In [32] decidability of validity and model-checking of a spatial logic describing trees *without* restricted names is studied. This logic is the quantifier-free static fragment of the Ambient Logic. Extensions of this logic can be used to describe [37], query [42], and reason about [56] tree-shaped semistructured data.

In this part of the Thesis we study decidability of validity, satisfiability, and model-checking for spatial logics describing trees (or static ambients) with restricted names (the expression "decidability of a logic" is used for "decidability of validity and satisfiability for closed formulas of that logic").

In particular we study how the introduction of freshness, revelation, and hiding influences decidability. While we started this work with the aim of proving decidability of hiding, we found out quite a different situation:

- freshness without revelation gives a rich decidable logic (Corollary 6.3.3)
- even a minimal logic (conjunction, negation, and binary relations) becomes undecidable if it is enriched with revelation (Corollary 7.4.1) or with hiding (Corollary 7.4.2).

Another contribution is the study of quantifier extrusion in SL. We introduce an extrusion algorithm for freshness (Lemma 6.2.1), and we prove that no extrusion algorithm exists for first order quantifiers, revelation, and hiding (Corollary 6.3.4).

5.1 Abstract Tree Model

We study logics that describe trees labeled with public and restricted names. Here we define the data model.

Definition 5.1.1. *The set $\mathcal{T}_\Lambda$ of the abstract trees generated by an infinite name set Λ is defined by the following grammar, with $n \in \Lambda$.*

$$\begin{array}{llll} \textit{trees} & T, U ::= & \mathbf{0} & \textit{empty tree} \\ & & n[T] & \textit{tree branch} \\ & & T \mid U & \textit{composition of trees} \\ & & (\nu n)\, T & \textit{restricted name} \end{array}$$

Free names $fn(T)$ and bound names are defined as usual. On these trees we define the usual congruence rules, with extrusion of restricted names. (Renaming) is the crucial rule, expressing the computational irrelevance of restricted names.

Table 5.1.1. *Congruence rules*

$T \equiv T$	(Refl)	$T \equiv U \Rightarrow n[T] \equiv n[U]$	(Amb)
$T \equiv U,\, U \equiv V \Rightarrow T \equiv V$	(Trans)	$T \equiv U \Rightarrow T \mid V \equiv U \mid V$	(Par)
$T \equiv U \Rightarrow U \equiv T$	(Symm)	$T \equiv U \Rightarrow (\nu n)\, T \equiv (\nu n)\, U$	(Res)
$T \mid \mathbf{0} \equiv T$	(Par Zero)	$T \mid U \equiv U \mid T$	(Par Comm)
$(T \mid U) \mid V \equiv T \mid (U \mid V)$	(Par Assoc)		
$m \notin fn(T) \Rightarrow (\nu n)\, T \equiv (\nu m)\, T\{n \leftarrow m\}$		(Renaming)	
$(\nu n)\, \mathbf{0} \equiv \mathbf{0}$		(Extr Zero)	
$n \notin fn(T) \Rightarrow T \mid (\nu n)\, U \equiv (\nu n)\, (T \mid U)$		(Extr Par)	
$n_1 \neq n_2 \Rightarrow n_1[(\nu n_2)\, T] \equiv (\nu n_2)\, n_1[T]$		(Extr Amb)	
$(\nu n_1)\, (\nu n_2)\, T \equiv (\nu n_2)\, (\nu n_1)\, T$		(Extr Res)	

Lemma 5.1.2 (Free Names)**.** *If $T \equiv U$ then $fn(T) = fn(U)$*

Lemma 5.1.3 (Inversion (see [44]))**.**

1. *If $(\nu n)\, T \equiv \mathbf{0}$ then $T \equiv \mathbf{0}$*
2. *If $(\nu n)\, T \equiv m[U]$ then $\exists U' \in \mathcal{T}_\Lambda.\ T \equiv m[U'],\, U \equiv (\nu n)\, U'$*
3. *If $(\nu n)\, T \equiv U \mid U'$ then $\exists \bar{U}, \bar{U}' \in \mathcal{T}_\Lambda.\ T \equiv \bar{U} \mid \bar{U}',\, \bar{U} \equiv (\nu n)\, U,\, \bar{U}' \equiv (\nu n)\, U'$*

Definition 5.1.4. *The set of trees in* extruded normal form *(ENF) is the least set such that:*

- *a tree with no local restriction is in ENF;*
- *if T is in ENF and $n \in fn(T)$ then $(\nu n)\, T$ is in ENF.*

Hence, a tree is in ENF iff it is composed by a prefix of restrictions followed by a restriction-free *matrix*, all the restricted names actually appear in the tree, and all the restricted names are mutually different.

Notation 5.1.5. *We will use ENF to denote the set of all terms in ENF, and $ENF(T)$ to denote the set $\{U : U \in ENF,\ U \equiv T\}$.*

Lemma 5.1.6. *For every tree T there exists U such that $T \equiv U$ and U is in ENF.*

The next lemma says that the ENF of a congruence class of trees is unique modulo renaming of bound names, reordering of the prefix, and congruence of the renamed matrix.

Lemma 5.1.7. *If $(\nu n_1) \ldots (\nu n_j)\, U \equiv (\nu n'_1) \ldots (\nu n'_k)\, U'$, and the two trees are in ENF, and U and U' are the matrixes, then $j = k$ and there exists a bijection τ between $\{n_1, \ldots, n_j\}$ and $\{n'_1, \ldots, n'_k\}$ such that*

$$\exists U''.\ U \equiv U'' = U'\{n'_i \leftarrow \tau(n_i)\}^{i \in 1..j}$$

5.2 Logic with Revelation and Quantifiers

5.2.1 Definition

We will study sublogics of the Ambient Logic without recursion and where no temporal operator appears. The logic is very rich, but we give here only a brief description. For more details see [28, 44, 45].

Definition 5.2.1. Spatial Logic formulas and satisfaction

$A, B ::= \mathbf{0}$	*empty tree*		
$\eta[A]$	*location*	$A@\eta$	*location adjunct*
$A \mid B$	*composition of trees*	$A \triangleright B$	*composition adjunct*
$A \wedge B$	*conjunction*	$\neg A$	*negation*
$\exists x.\, A$	*existential quantification*	$\mathsf{N}x.\, A$	*fresh quantification*
$\eta(\mathrm{R})A$	*revelation*	$A \oslash \eta$	*revelation adjunct*

$$
\begin{array}{lcl}
T \models \mathbf{0} & \triangleq & T \equiv \mathbf{0} \\
T \models n[A] & \triangleq & \exists U \in \mathcal{T}_\Lambda.\ T \equiv n[U] \text{ and } U \models A \\
T \models A@n & \triangleq & n[T] \models A \\
T \models A \mid B & \triangleq & \exists T_1, T_2 \in \mathcal{T}_\Lambda.\ T \equiv T_1 \mid T_2 \text{ and } T_1 \models A \text{ and } T_2 \models B \\
T \models A \triangleright B & \triangleq & \forall U \in \mathcal{T}_\Lambda.\ U \models A \text{ implies } T \mid U \models B \\
T \models A \wedge B & \triangleq & T \models A \text{ and } T \models B \\
T \models \neg A & \triangleq & T \not\models A \\
T \models \mathsf{N}x.\, A & \triangleq & \exists n \notin (fn(T) \cup nm(A)).\ T \models A\{x \leftarrow n\} \\
T \models \exists x.\, A & \triangleq & \exists n \in \Lambda.\ T \models A\{x \leftarrow n\} \\
T \models n(\mathrm{R})A & \triangleq & \exists U \in \mathcal{T}_\Lambda.\ T \equiv (\nu n)\, U \text{ and } U \models A \\
T \models A \oslash n & \triangleq & (\nu n)\, T \models A
\end{array}
$$

The set $\mathcal{A}$ of the formulas of the full logic is defined by the grammar shown in Table 5.2.5.2.1 (we will consider some sub-logics later on). η stands for either a name $n \in \Lambda$ or a name variable $x \in \mathcal{X}$. In Table 5.2.5.2.1 we also define the satisfaction of a closed formula A by a model T ($T \models A$). We use $nm(A)$ to denote the set of all names n that appear in a formula.

We will also use $T, \rho \models A$, where ρ is a ground substitution mapping $fv(A)$ into Λ, as an alternative notation for $T \models A\rho$, where $A\rho$ is the closed formula obtained by applying ρ to all of its free variables.

Notation 5.2.2. *$SL_{\{\}}$ will denote the logic fragment without quantifiers, revelation and revelation adjunct. SL_X will denote the extension of $SL_{\{\}}$ with the logical operators in X. Hence the full logic of Definition 5.2.1 is $SL_{\{\mathsf{N}, (\mathrm{R}), \exists, \oslash\}}$.*

We define $fn(T, A) \triangleq fn(T) \cup nm(A)$. We assume that $\exists x$, $\mathsf{N}x$ and $\eta(\mathrm{R})$ bind as

far to the right as possible, so that, for example, $\exists x.\, A \wedge \exists y.\, B$ is the same as $\exists x.\, (A \wedge \exists y.\, B)$. We assume the usual definitions for: (i) the derived operators $A \vee B$, $\mathbf{T}$, $\mathbf{F}$, $\forall x.A$, $\eta \neq \eta'$, $A \Rightarrow B$, $A \Leftrightarrow B$; (ii) free variables $fv(A)$. It is worth emphasizing that revelation is not a binder, i.e. $fv(\eta \mathrel{Ⓡ} A) = fv(\eta) \cup fv(A)$. $fv(\eta)$ is defined as $\{\eta\}$ when η is a variable x, and as $\emptyset$ when η is a name n. Closed formulas are formulas without free variables.

We will also study the properties of the following derived operators:

operator		*definition*	*fundamental property (may be used as a definition)*
$\mathbf{H}x.\, A$	$\triangleq$	$\text{И}x.\, x \mathrel{Ⓡ} A$	$T \models \mathbf{H}x.\, A \;\Leftrightarrow\; \exists n \notin nm(A).\, \exists U \in \mathcal{T}_\Lambda.\, T \equiv (\nu n)\, U,\, U \models A\{x \leftarrow n\}$
$Ⓒ\, n$	$\triangleq$	$\neg n \mathrel{Ⓡ} \mathbf{T}$	$T \models Ⓒ\, n \;\Leftrightarrow\; n \in fn(T)$
$n = m$	$\triangleq$	$(n[\mathbf{T}]) @ m$	$T \models n = m \;\Leftrightarrow\; n = m$

In a nutshell, the structural operators $\mathbf{0}$, $\eta[A]$, $A \mid B$, allow one to explore the structure of the model, so that $T \models n[(m[\mathbf{T}] \vee p[\mathbf{0}])]$ specifies that T matches either $n[m[U]]$ or $n[p[\mathbf{0}]]$. The adjunct operators $@$, $\triangleright$, $\oslash$, describe how the model behaves when it is inserted into a context $n[\_], U \mid \_$, or $(\nu n)\, \_ .\, \triangleright$ is very expressive, since it can be used to reduce validity to model-checking. Consider now a tree $T \equiv (\nu p)\, m[p[\mathbf{0}]]$ with a restricted name. This can be described by the formula $n \mathrel{Ⓡ} m[n[\mathbf{T}]]$, which uses n to talk about the "anonymous" p:

$$(\nu p)\, m[p[\mathbf{0}]] \models n \mathrel{Ⓡ} m[n[\mathbf{T}]] \;\Leftrightarrow\; (\nu p)\, m[p[\mathbf{0}]] \equiv (\nu n)\, m[n[\mathbf{0}]],\; m[n[\mathbf{0}]] \models m[n[\mathbf{T}]]$$

However, the satisfaction of this formula depends upon the specific name n: $T \models n \mathrel{Ⓡ} n[\mathbf{T}]$, literally means that $T \equiv (\nu n)\, n[U]$ for some U, which is satisfied by any $(\nu p)\, p[U]$, unless n happens to be free in $(\nu p)\, p[U]$ (in this case, $(\nu p)\, p[U] \not\equiv (\nu n)\, n[U]$). In many situations, we really want to say things like 'T has a shape $(\nu x)\, x[U]$' where no name should be prevented from matching x by the irrelevant fact that it appears free in T. To this aim, we must use a name that is guaranteed to be fresh, which can be obtained through Gabbay-Pitts fresh name quantification: $\text{И}x.\, x \mathrel{Ⓡ} x[\mathbf{T}]$. The И-Ⓡ jargon is encoded by hiding quantification: $\text{И}x.\, x \mathrel{Ⓡ} x[\mathbf{T}] \stackrel{def}{=} \mathbf{H}x.\, x[\mathbf{T}]$.

$\mathbf{H}$ may be taken as primitive instead of И and Ⓡ, but one would lose (in a logic without adjuncts) the ability to express the property $Ⓒ\, \eta$. Hence, one would consider the pair $\mathbf{H}$-ⓒ as an alternative to И-Ⓡ. This motivated us to study the decidability properties of all these operators. The result is symmetric: each pair contains one operator ($\mathbf{H}$/Ⓡ) which is undecidable even when confined to a tiny sublogic, and an operator which we prove to be decidable (ⓒ/И); ⓒ and И are even decidable together. (We prefer the canonical choice of И-Ⓡ because we find their definitions more elegant, and since the encoding of the other two operators is very direct; the reverse encoding is much harder.)

$\mathbf{H}x.\, A$ is quite similar to an existential quantification over the names that are restricted in the model, but there are some subtleties. For example, two different

hiding-quantified variables cannot be bound to the same restricted name, i.e., while $n[n[\mathbf{0}]] \models \exists x.\ \exists y.\ x[y[\mathbf{0}]]$, $(\nu n)\, n[n[\mathbf{0}]] \not\models \mathbf{H}x.\, \mathbf{H}y.\, x[y[\mathbf{0}]]$: after x is bound to n, n is not restricted any more, hence y cannot be bound to n.

Hiding, freshness, appearance ($\textcircled{c}$), and revelation can be used to express essential properties in any specialization of this logic to specific computational structures. We present here some examples in a very informal way, just to give the flavour of the applications of the hiding operator.

When restricted names are used to represent pointers, the presence of a dangling pointer can be formalized as follows [38]; here $.n[A]$ abbreviates $n[A] \mid \mathbf{T}$, hence means: there is a branch $n[U]$ that satisfies $n[A]$.

$$\mathbf{H}x.\, (.\mathit{paper}[.\mathit{citing}[x]] \wedge \neg .\mathit{paper}[.\mathit{paperId}[x]])$$

If restricted names represent passwords in a concurrent system (e.g. in [28]), we can specify properties like 'inside k we find a password which will not be communicated', with the following sentence, where '$\Diamond A$' means 'in some process deriving from the current process A holds', and '$\mathit{send}(m, n)$' means 'm is ready for transmission on a channel n'.

$$\mathbf{H}x.\, .k[x] \wedge \neg\exists n.\, \Diamond \mathit{send}(x, n)$$

If restricted names represent α-renamable variable names, the following sentence describes any tree that represents a lambda term; $\mu X.A$ is a recursive definition, where each occurrence of X can be expanded with the body A. It says: a lambda term is either a free variable, or an application, or a lambda binder that pairs an α-renamable name with a body, where that name may appear free. The interplay between μ and $\mathbf{H}$ ensures that no variable appears twice in the same scope.

$$\mu LT.\ (\exists x.\, \mathit{var}[x]) \vee (\mathit{function}[LT] \mid \mathit{argument}[LT]) \vee (\mathbf{H}x.\, \mathit{lambda}[x] \mid \mathit{body}[LT])$$

We now define the standard notions of formula validity, satisfiability, of formula implication, and of formula equivalence for spatial logics.

$$\begin{array}{lcll}
\mathbf{vld}(A) & \triangleq & \forall T \in \mathcal{T}_\Lambda.\ \forall \rho : \mathit{fv}(A) \to \Lambda.\ T, \rho \models A & \textit{(validity)} \\
\mathbf{sat}(A) & \triangleq & \exists T \in \mathcal{T}_\Lambda.\ \exists \rho : \mathit{fv}(A) \to \Lambda.\ T, \rho \models A & \textit{(satisfiability)} \\
A \vdash B & \triangleq & \forall T \in \mathcal{T}_\Lambda.\ \forall \rho : (\mathit{fv}(A) \cup \mathit{fv}(B)) \to \Lambda. & \\
 & & \qquad T, \rho \models A \ \Rightarrow\ T, \rho \models B & \textit{(implication)} \\
A \dashv\vdash B & \triangleq & A \vdash B \text{ and } B \vdash A & \textit{(equivalence)}
\end{array}$$

Let $\vec{\forall} A$ denote $\forall x_1 \ldots \forall x_n.\ A$, where $\{x_1 \ldots x_n\} = \mathit{fv}(A)$, and similarly for $\vec{\exists} A$. The following properties come from [44, 32], or are easily derivable from there.

Table 5.2.2. *Properties of SL*

(Implication)	$A \vdash B \ \Leftrightarrow\ \mathbf{vld}(A \Rightarrow B)$		$A \dashv\vdash B \ \Leftrightarrow\ \mathbf{vld}(A \Leftrightarrow B)$
(Closure)	$\mathbf{vld}(A) \ \Leftrightarrow\ \mathbf{vld}(\vec{\forall} A)$		$\mathbf{sat}(A) \ \Leftrightarrow\ \mathbf{sat}(\vec{\exists} A)$
($\mathbf{vld}$ by $\models$)	$\mathbf{vld}(A) \ \Leftrightarrow\ \mathbf{0} \models \mathbf{T} \triangleright \vec{\forall} A$	$\Leftrightarrow$	$\mathbf{0} \models \vec{\forall}(\mathbf{T} \triangleright A)$

The last property shows how validity can be reduced to model-checking using $\triangleright$ and quantification, or just $\triangleright$ alone, when the formula is closed [32].

Definition 5.2.3 (Formula with Holes (see [44]))**.** *We use $B\{-\}$ to indicate a formula with a set of formula holes, indicated by $-$, and $B\{A\}$ to denote the formula obtained by filling these holes with A, after renaming the variables in B to avoid capturing variables of A.*

Lemma 5.2.4 (Substitution (see [44]))**.**

$$\mathbf{vld}(A \Leftrightarrow A') \Rightarrow \mathbf{vld}(B\{A\} \Leftrightarrow B\{A'\}) \quad i.e. A \dashv\vdash A' \Rightarrow B\{A\} \dashv\vdash B\{A'\}$$

We introduce a couple of lemmas and corollaries which will be useful in Chapter 7.

Lemma 5.2.5. *Let $(\vec{\nu}_{i\in I} n_i)\, T$ be in ENF, let T be the matrix, and let m be a name not in $\{n_i\}^{i\in I}$; then:*

$$(\vec{\nu}_{i\in I} n_i)\, T \models m \textcircled{\scriptsize R} A \;\Leftrightarrow\; m \notin fn(T) \wedge ((\exists h \in I.\ (\vec{\nu}_{i\in(I\setminus\{h\})} n_i)\, T\{n_h \leftarrow m\} \models A) \vee (\vec{\nu}_{i\in I} n_i)\, T \models A)$$

Corollary 5.2.6. *For $(\vec{\nu}_{i\in I} n_i)\, T$ in ENF, where T is the matrix, if*

$$\forall n \in fn(T).\ \exists T'.\ T \equiv n[\mathbf{0}] \mid T'$$

then:

$$\begin{aligned}&(\vec{\nu}_{i\in I} n_i)\, T \models m \textcircled{\scriptsize R} (A \wedge (m[\mathbf{0}] \mid \mathbf{T}))\\ &\Leftrightarrow\; m \notin fn(T) \wedge \exists h \in I.\ (\vec{\nu}_{i\in(I\setminus\{h\})} n_i)\, T\{n_h \leftarrow m\} \models A\end{aligned}$$

Lemma 5.2.7. *For $(\vec{\nu}_{i\in I} n_i)\, T$ in ENF, where T is the matrix:*

$$\begin{aligned}&(\vec{\nu}_{i\in I} n_i)\, T \models \mathbf{H}x.\, A\\ &\quad\Leftrightarrow\; (\forall m \notin (nm(A) \cup fn(T)).\\ &\qquad \exists h \in I.\ (\vec{\nu}_{i\in(I\setminus\{h\})} n_i)\, T\{n_h \leftarrow m\} \models A\{x \leftarrow m\})\\ &\qquad \vee\ (\vec{\nu}_{i\in I} n_i)\, T \models A\\ &\quad\Leftrightarrow\; (\exists m \notin (nm(A) \cup fn(T)).\\ &\qquad \exists h \in I.\ (\vec{\nu}_{i\in(I\setminus\{h\})} n_i)\, T\{n_h \leftarrow m\} \models A\{x \leftarrow m\})\\ &\qquad \vee\ (\vec{\nu}_{i\in I} n_i)\, T \models A\end{aligned}$$

Corollary 5.2.8. *For $(\vec{\nu}_{i\in I} n_i)\, T$ in ENF, where T is the matrix, if*

$$\forall n \in fn(T).\ \exists T'.\ T \equiv n[\mathbf{0}] \mid T'$$

then:

$$\begin{aligned}&(\vec{\nu}_{i\in I} n_i)\, T \models \mathbf{H}x.\, A \wedge (x[\mathbf{0}] \mid \mathbf{T})\\ &\quad\Leftrightarrow\; \forall m \notin (nm(A) \cup fn(T)).\\ &\qquad \exists h \in I.\ (\vec{\nu}_{i\in(I\setminus\{h\})} n_i)\, T\{n_h \leftarrow m\} \models A\{x \leftarrow m\}\\ &\quad\Leftrightarrow\; \exists m \notin (nm(A) \cup fn(T)).\\ &\qquad \exists h \in I.\ (\vec{\nu}_{i\in(I\setminus\{h\})} n_i)\, T\{n_h \leftarrow m\} \models A\{x \leftarrow m\}\end{aligned}$$

Lemma 5.2.9 (Satisfaction is up to $\equiv$)**.** *If $T \models A$ and $T \equiv U$ then $U \models A$*

Lemma 5.2.10. *If $m \notin fn(T, A)$ and $n \notin fn(T, A)$ then:*

$$T \models A\{x \leftarrow m\} \Leftrightarrow T \models A\{x \leftarrow n\}$$

Corollary 5.2.11. *If $\rho, \rho' : \mathcal{X} \xrightarrow{in} \Lambda$, $\rho\downarrow = \rho'\downarrow$, $\rho\uparrow \# fn(T, A)$, and $\rho'\uparrow \# fn(T, A)$, then:*

$$T, \rho \models A \;\Leftrightarrow\; T, \rho' \models A$$

The next Corollary will be used often in the following chapters: we will use each of (1)-(4) as if it were the definition of (5). From now on, every quantification on sets on names will always be implicitly (or explicitly) qualified to range over finite sets of names only.

Corollary 5.2.12 (Gabbay-Pitts Property)**.** *For any $\mathbf{N} \subset \Lambda$ finite, all the following are equivalent:*

1. $\forall m \notin fn(T, A).\ T \models A\{x \leftarrow m\}$
2. $\forall m \notin (fn(T, A) \cup \mathbf{N}).\ T \models A\{x \leftarrow m\}$
3. $\exists m \notin (fn(T, A) \cup \mathbf{N}).\ T \models A\{x \leftarrow m\}$
4. $\exists m \notin fn(T, A).\ T \models A\{x \leftarrow m\}$
5. $T \models \mathsf{N}x.\, A$

Proof. (1) $\Rightarrow$ (2): $m \notin (fn(T, A) \cup \mathbf{N}) \Rightarrow m \notin fn(T, A)$.
(2) $\Rightarrow$ (3): $\Lambda \setminus (fn(T, A) \cup \mathbf{N})$ is not empty, since $fn(T, A) \cup \mathbf{N}$ is finite.
(3) $\Rightarrow$ (4): $m \notin (fn(T, A) \cup \mathbf{N}) \Rightarrow m \notin fn(T, A)$.
(4) $\Rightarrow$ (1): by Lemma 5.2.10
(4) $\Leftrightarrow$ (5): by Definition □

Chapter 6

Decidability of Freshness

In this Chapter we prove decidability of $SL_{\{\mathbb{O},\mathcal{O}\}}$ and we extend the result to $SL_{\{\mathbb{O},\mathcal{O},\mathsf{N}\}}$ using an extrusion algorithm for freshness quantification.

An extrusion algorithm for a set of logical operators O is an algorithm that transforms a formula into an equivalent formula in O-prenex form, i.e. into a formula formed by a prefix of operators from O followed by a matrix where they do not appear (i.e. first order logic admits a simple extrusion algorithm for the pair $\exists, \forall$). In the following we will show that:

- in a spatial logic with the $\triangleright$ operator, extrusion implies decidability (Corollary 6.3.2);
- the freshness quantifier admits extrusion (Lemma 6.2.1), hence is decidable;
- undecidability of the revelation operator, existential quantifier, and hiding quantifier, implies that no extrusion algorithm can exist for them (Corollary 6.3.4).

6.1 Extending the STL result to Abstract trees

We start from the following result presented in [32].

Theorem 6.1.1 (Calcagno-Cardelli-Gordon)**.** *The model-checking problem restricted to closed formulas with no quantification and revelation is decidable over trees with no local names.*

Note that (as shown in [32]) in SL fragments with composition adjunct model-checking and validity problem are equivalent. Thus, in these cases decidability of model checking implies decidability of validity.

Corollary 6.1.2 (Calcagno-Cardelli-Gordon)**.** *The validity and satisfiability problems restricted to closed formulas with no quantification and revelation are decidable over trees with no local names.*

We will extend this result by adding restricted names to the models and the revelation adjunct ($A \oslash n$) to the logic. We will follow the schema of [32] to prove decidability of the resulting logic. The idea is to find an equivalence relation between abstract trees not distinguishable by formulas of the same size (the logical equivalence). In [32] the idea was that a formula can distinguish up to a depth h and a width w.

6.1.1 Logical Equivalence

In our case the formulas can also: (i) distinguish trees with different free names (ii) count sub-trees (up to the width w) with the same set of free names, in particular these trees can be "sealed". We say that a tree T is "sealed" if it is only congruent to restricted terms (modulo neutral element), i.e.

$$T \text{ is sealed} \quad \Leftrightarrow \quad \forall U.\, U \equiv T \Rightarrow \exists n, U'.\, U \approx (\nu n)\, U'.$$

Where $\approx$ is the smallest equivalence relation with associativity, commuativity and neutral element of composition (that is U is a revelation plus some empty trees). For example, $(\nu n)\,(n[\mathbf{0}] \mid m[n])$ is sealed while $(\nu n)\,(\nu m)\,(n[\mathbf{0}] \mid m[\mathbf{0}])$ is not.

Essentially a formula without revelation cannot observe inside sealed trees, but it can infer (using the revelation adjunct) the set of free variables inside it. In general we will see that the appearence can be derived from revelation adjunct, so a formula with N as free names can separate the free names of the models up to N. For this reasons we define $fn_N(T) = fn(T) \cap N$ and the following equivalence relations:

Definition 6.1.3 (Relation $\cong_{w,N}$)**.**

$$\begin{array}{l} T \cong_{w,N} U \quad \Leftrightarrow \\ \quad \forall i \in 1..w,\ T_j \not\equiv \mathbf{0} \text{ with } j \in 1..i \\ \quad\quad \text{if } T \equiv T_1 \mid \ldots \mid T_i \\ \quad\quad \text{then } U \equiv U_1 \mid \ldots \mid U_i \\ \quad\quad \text{with } U_j \not\equiv \mathbf{0} \text{ and } fn_N(T_j) = fn_N(U_j) \text{ for } j \in 1..i \\ \quad\quad \text{and vice versa} \end{array}$$

Definition 6.1.4 (Relation $\sim_{h,w,N}$)**.**

$$\begin{array}{l} T \sim_{0,w,N} U \quad \Leftrightarrow T \cong_{w,N} U \\ T \sim_{h+1,w,N} U \Leftrightarrow T \cong_{w,N} U \text{ and} \\ \quad \forall i \in 1..w,\ n \in N,\ T_j \text{ with } j \in 1..i \\ \quad\quad \text{if } T \equiv n[T_1] \mid \ldots \mid n[T_i] \mid T' \\ \quad\quad \text{then } U \equiv n[U_1] \mid \ldots \mid n[U_i] \mid U' \end{array}$$

such that $T_j \sim_{h,w,N} U_j$ *for* $j \in 1..i$
and vice versa

Note that $\sim_{h,w,N}$ is an equivalence relation: reflexivity, symmetry, and transitivity are immediate consequences of the definition. Moreover, it is preserved by congruence on trees:

Lemma 6.1.5. *If* $T \sim_{h,w,N} U$ *and* $U \equiv U'$ *then* $T \sim_{h,w,N} U'$

Proof. By easy induction on h. □

Lemma 6.1.6. *When* $w > 0$ *if* $T \sim_{h,w,N} U$ *then* $fn_N(T) = fn_N(U)$

Proof. Immediate taking $i = 1$. □

Lemma 6.1.7. *If* $T \sim_{h,w,N} U$ *and* $h' \leq h, w' \leq w, N' \subseteq N$ *then* $T \sim_{h',w',N'} U$

Proof. By induction on h observing that if $fn_N(T) = fn_N(U)$ and $N' \subseteq N$ then $fn_{N'}(T) = fn_{N'}(U)$ □

We need an entaliment of the inversion lemma:

Lemma 6.1.8. *If* $(\nu n)\, T \equiv T_1 \mid \ldots \mid T_k$ *then there exists a* $j \in 1..k$ *and a tree* U *such that* $(\nu n)\, U \equiv T_j$ *and* $T \equiv T_1 \mid \ldots \mid T_{j-1} \mid U \mid T_{j+1} \mid \ldots \mid T_k$

Proof. We prove it by cases: if $n \notin fn(T)$ then the result is immediate. Suppose that $n \in fn(T)$, take the normal form of $T \equiv (\nu\vec{\gamma})\, T'$ with $n \notin \vec{\gamma}$, then the ENF of $(\nu n)\, T$ is of the form $(\nu n\vec{\gamma})\, T'$. Take the normal forms of $T_i \equiv (\nu\vec{\alpha_i})\, T_i'$ such that all $\vec{\alpha_i}$ are disjoint and $n \notin \alpha_i$. We have (by extrusion) that the normal form of $T_1 \mid \ldots \mid T_k$ is $(\nu\vec{\alpha_1} \ldots \vec{\alpha_k})\, T_1' \mid \ldots \mid T_k'$. Since $T_1 \mid \ldots \mid T_k \equiv (\nu n)\, T$, then their normal forms must be the same modulo renaming and permutations. Thus, there must exist a permutation τ such that $T' \equiv (T_1' \mid \ldots \mid T_k')\{n\vec{\gamma} \leftarrow \tau(\vec{\alpha_1} \ldots \vec{\alpha_k})\}$. Take the j such that $n \in \tau(\vec{\alpha_j})$, then take $m = \tau^{-1}(n)$ and $U = (\nu\vec{\alpha_j} \setminus m)\, T_j'\{n \leftarrow m\}$, so $(\nu n)\, U \equiv (\nu n)\, (\nu\vec{\alpha_j} \setminus m)\, T_j'\{n \leftarrow m\} \equiv (\nu m)\, (\nu\vec{\alpha_j} \setminus m)\, T_j' \equiv (\nu\vec{\alpha_j})\, T_j' \equiv T_j$. We have

$$
\begin{array}{l}
T_1 \mid \ldots \mid T_{j-1} \mid U \mid T_{j+1} \mid \ldots \mid T_k \equiv \\
(\nu\vec{\alpha_1})\, T_1' \mid \ldots \mid (\nu\vec{\alpha_j} \setminus m)\, T_j'\{n \leftarrow m\} \mid (\nu\vec{\alpha}_{j+1})\, T_{j+1}' \mid \ldots \mid (\nu\vec{\alpha_k})\, T_k' \equiv \\
(\nu\vec{\alpha_1} \ldots \vec{\alpha_k} \setminus m)\, (T_1' \mid \ldots \mid T_k')\{n \leftarrow m\} \equiv \\
(\nu\vec{\gamma})\, (T_1' \mid \ldots \mid T_k')\{n \leftarrow m\}\{\vec{\gamma} \leftarrow \tau(\vec{\alpha_1} \ldots \vec{\alpha_k} \setminus m)\} \equiv \\
(\nu\vec{\gamma})\, (T_1' \mid \ldots \mid T_k')\{n\vec{\gamma} \leftarrow \tau(\vec{\alpha_1} \ldots \vec{\alpha_k})\} \equiv \\
(\nu\vec{\gamma})\, T' \equiv T
\end{array}
$$

□

We show that $\sim_{h,w,N}$ is a congruence:

Lemma 6.1.9 (Congruence)**.** *When $n \in N$ the following holds:*

$$\begin{array}{lll} \textbf{(1)} & T \sim_{h,w,N} U & \Rightarrow \; n[T] \sim_{h+1,w,N} n[U] \\ \textbf{(2)} & T \sim_{h,w,N} U & \Rightarrow \; (\nu n)\, T \sim_{h,w,N} (\nu n)\, U \\ \textbf{(3)} & T \sim_{h,w,N} U,\ T' \sim_{h,w,N} U' & \Rightarrow \; T \mid T' \sim_{h,w,N} U \mid U' \end{array}$$

Proof. We prove both parts directly.

(1) We prove first that $n[T] \cong_{w,N} n[U]$. Consider any $i \in 1..w$, $n \in N$, $T_j \not\equiv \mathbf{0}$ for $j \in 0..i$ such that

$$n[T] \equiv T_0 \mid ... \mid T_i$$

Then $i = 1$ and $n[T] \equiv T_1$. Take $U_1 = n[U]$ with $U_1 \not\equiv \mathbf{0}$ and $fn_N(T_1) = fn_N(n[T]) = fn_N(n[U]) = fn_N(U_1)$. This proves the first equivalence (for every h).

For the second equivalence we follow the proof in [32], we report it here for completeness. Suppose $T \sim_{h,w,N} U$. We proceed by induction on h. If $h = 0$ then the conclusion follows by $n[T] \cong_{w,N} n[U]$. For $h+1$ consider any $i \in 1..w$, $m \in N$, T_j with $j \in 1..i$ such that

$$n[T] \equiv m[T_1] \mid ... \mid m[T_i] \mid T'$$

Then $i = 1$ and $n = m$ and $T \equiv T_1$ and $T' \equiv \mathbf{0}$. We have $n[U] \equiv n[U] \mid \mathbf{0}$, and $T_1 \sim_{h,w,N} U$ by Lemma 6.1.5. This proves $n[T] \sim_{h+1,w,N} n[U]$.

(2) We prove first that $(\nu n)\, T \cong_{w,N} (\nu n)\, U$. Consider any $i \in 1..w$, $n \in N$, $T_j \not\equiv \mathbf{0}$ for $j \in 1..i$ such that

$$(\nu n)\, T \equiv T_1 \mid ... \mid T_i$$

By Lemma 6.1.8 there exist $\bar{T}_k$ such that $T \equiv T_1 \mid ... \mid \bar{T}_k \mid ... \mid T_i$ and $T_k \equiv (\nu n)\, \bar{T}_k$. By $T \sim_{h+1,w,N} U$ there exist also $U_1, ..., U_k, ..., U_i$ all not congruent to $\mathbf{0}$ such that $U \equiv U_1 \mid ... \mid \bar{U}_k \mid ... \mid U_i$ with $fn_N(T_j) = fn_N(U_j)$ and $fn_N(\bar{T}_k) = fn_N(\bar{U}_k)$. By congruence, $(\nu n)\, U \equiv (\nu n)\, (U_1 \mid ... \mid \bar{U}_k \mid ... \mid U_i)$ and, since $n \in N$ and $n \notin fn_N(U_j)$ implies $n \notin fn(U_j)$, we can intrude abstraction (apart from the term $\bar{U}_k$). Thus $(\nu n)\, U \equiv U_1 \mid ... \mid (\nu n)\, \bar{U}_k \mid ... \mid U_i$ and $fn_N((\nu n)\, \bar{U}_k) = fn_N(\bar{U}_k) \setminus \{n\} = fn_N(\bar{T}_k) \setminus \{n\} = fn_N(T_k)$. This proves the first equivalence (for every h).

For the second equivalence we proceed by induction on h. If $h = 0$ then the conclusion follows by $(\nu n)\, T \cong_{w,N} (\nu n)\, U$. For $h + 1$ suppose $T \sim_{h+1,w,N} U$ and consider any $i \in 1..w$, $m \in N$, T_j for $j \in 1..i$ such that

$$(\nu n)\, T \equiv m[T_1] \mid ... \mid m[T_i] \mid T_{i+1} \qquad (\phi_1)$$

Observe that $n \notin fn(m[T_1] \mid ... \mid m[T_i] \mid T_{i+1})$ since $n \notin fn((\nu n)\, T)$, thus $n \neq m$ and $n \notin fn(T_j)$ for $j \in 1..i + 1$.

By Lemma 6.1.8 and extrusion on ϕ_1 there exist k and $\bar{T_k}$ such that

$$T \equiv m[T_1] \mid ... \mid m[\bar{T_k}] \mid ... \mid T_{i+1}$$

and $T_k \equiv (\nu n)\, \bar{T_k}$ with $k \leq i+1$.

Since $T \sim_{h+1,w,N} U$ there exist also $U_1, ..., \bar{U_k}, ..., U_{i+1}$ such that

$$U \equiv m[U_1] \mid ... \mid m[\bar{U_k}] \mid ... \mid U_{i+1} \qquad (\phi_2)$$

and $\bar{T_k} \sim_{h,w,N} \bar{U_k}$ and $T_j \sim_{h,w,N} U_j$ for $j \leq i+1, j \neq k$. By induction $\bar{T_k} \sim_{h,w,N} \bar{U_k} \Rightarrow (\nu n)\, \bar{T_k} \sim_{h,w,N} (\nu n)\, \bar{U_k}$, so by Lemma 6.1.5 we have $T_k \sim_{h,w,N} (\nu n)\, \bar{U_k}$.

By congruence on ϕ_2 we have $(\nu n)\, U \equiv (\nu n)\, (m[U_1] \mid \ldots \mid m[\bar{U_k}] \mid ... \mid U_{i+1})$. Since $n \notin fn(T_j)$ and $fn_N(T_j) = fn_N(U_j)$ (by Lemma 6.1.6), by extrusion and $n \neq m$ we have $(\nu n)\, U \equiv m[U_1] \mid \ldots \mid m[(\nu n)\, \bar{U_i}] \mid (\nu n)\, U_{i+1}$ and we obtain the equivalence with $U_i = (\nu n)\, \bar{U_i}$.

(3) We prove first that $(\nu n)\, T \mid T' \cong_{w,N} (\nu n)\, U \mid U'$. Consider any $i \in 1..w$, $V_j \not\equiv \mathbf{0}$ with for $j \in 1..i$ such that

$$T \mid U \equiv V_1 \mid ... \mid V_i$$

Suppose without loss of generality that the V_j are ordered in a way that there exists $k \in 1..i$ such that

$$T \equiv V_1 \mid ... \mid V_k \quad U \equiv V_{k+1} \mid ... \mid V_i$$

Since $k \in 1..w$, from $T \sim_{h,w,N} T'$ we have

$$T' \equiv T'_1 \mid ... \mid T'_k \text{ such that } T'_j \not\equiv \mathbf{0} \text{ and } fn_N(V_j) = fn_N(T'_j) \text{ for } j \in 1..k$$

Similarly, from $U \sim_{h,w,N} U'$ we have

$$U' \equiv U'_{k+1} \mid ... \mid U'_i \text{ such that } U'_j \not\equiv \mathbf{0} \text{ and } fn_N(V_j) = fn_N(U'_j) \text{ for } j \in (k+1)..i$$

Hence, we have

$$T' \mid U' \equiv T'_0 \mid ... \mid T'_k \mid U'_{k+1} \mid ... \mid U'_i$$

Since $fn_N(V_j) = fn_N(T'_j)$ for $j \in 1..k$ and $fn_N(V_j) = fn_N(U'_j)$ for $j \in (k+1)..i$, this proves that for each h. $T \mid U \sim_{h+1,w,N} T' \mid U'$

For the second equivalence we proceed as in [32], by induction on h. If $h = 0$ then the conclusion is immediate. For $h+1$, suppose $T \sim_{h+1,w,N} U$ and $T' \sim_{h+1,w,N} U'$; then consider any $i \in 1..w$, $n \in N$, V_j for $j \in 1..i$ such that

$$T \mid U \equiv m[V_0] \mid ... \mid m[V_i] \mid V_{i+1}$$

Suppose without loss of generality that the V_j are ordered in a way that there exists $k \in 1..i$, $\bar{T}$,$\bar{U}$ such that

$$T \equiv n[V_0] \mid ... \mid n[V_k] \mid \bar{T} \quad U \equiv n[V_{k+1}] \mid ... \mid n[V_i] \mid \bar{U} \quad V_{i+1} \equiv \bar{T} \mid \bar{U}$$

Since $k \in 0..w$, from $T \sim_{h+1,w,N} T'$ we have

$$T' \equiv n[T'_0] \mid ... \mid n[T'_k] \mid \bar{T'} \text{ such that } V_j \sim_{h,w,N} T'_j \text{ for } j \in 0..k$$

Similarly, from $U \sim_{h+1,w,N} U'$ we have

$$U' \equiv n[U'_{k+1}] \mid ... \mid n[U'_i] \mid \bar{U'} \text{ such that } V_j \sim_{h,w,N} U'_j \text{ for } j \in (k+1)..i$$

Hence, we have

$$T' \mid U' \equiv n[T'_0] \mid ... \mid n[T'_k] \mid n[U'_{k+1}] \mid ... \mid n[U'_i] \mid \bar{T'} \mid \bar{U'}$$

Since $V_j \sim_{h,w,N} U'_j$ for $j \in 1..k$ and $V_j \sim_{h,w,N} U'_j$ for $j \in (k+1)..i$, this proves that $T \mid U \sim_{h+1,w,N} T' \mid U'$.

□

Lemma 6.1.10 (Inversion)**.** *If* $T_1 \mid T_2 \sim_{h,w_1+w_2,N} U$ *then*
$\exists U_1, U_2.\, U \equiv U_1 \mid U_2 \wedge U_1 \sim_{h,w_1,N} T_1 \wedge U_2 \sim_{h,w_2,N} T_2$

Proof. We proceed as in [32], but we use a normal form that takes into account also sealed trees. In particular we say that a tree T is in (h, w, N) normal form if whenever $T \equiv n[T_1] \mid n[T_2] \mid T'$ with $n \in N$ if $T_1 \sim_{h,w,N} T_2$ then $T_1 \equiv T_2$ and also if $T \equiv T_1 \mid T_2 \mid T'$ with T_1 sealed and T_2 sealed, if $fn_N(T_1) = fn_N(T_2)$ then $T_1 \equiv T_2$. We can construct the $(h+1, w, N)$ normal form easily by substituting T_2 with T_1 each time one of the two conditions is not valid (observing that with this substitution we obtain a tree that is congruent upto $h+1$). Thus we can write T_1, T_2, U in this normal form without loss of generality. Hence, there exist $s, k \leq s, T_j, n_j, a'_j, a''_j, b_j$ for $j \in 1..s$ such that

$$\begin{array}{lcl} T_1 & \equiv & a'_1 \cdot n_1[T_1] \mid ... \mid a'_k \cdot n_k[T_k] \mid a'_{k+1} \cdot T_{k+1} \mid ... \mid a'_s \cdot T_s \\ T_2 & \equiv & a''_1 \cdot n_1[T_1] \mid ... \mid a''_k \cdot n_k[T_k] \mid a''_{k+1} \cdot T_{k+1} \mid ... \mid a''_s \cdot T_s \\ U & \equiv & b_1 \cdot n_1[T_1] \mid ... \mid b_k \cdot n_k[T_k] \mid b_{k+1} \cdot T_{k+1} \mid ... \mid b_s \cdot T_s \end{array}$$

with $\forall i, j. \in 1..k.\, T_j \sim_{h,w,N} T_i \wedge n_i \neq n_j \Rightarrow i \neq j$ and $\forall i, j. \in k+1..s.\, fn_N(T_j) = fn_N(T_i) \Rightarrow i \neq j$. As in [32] we can specify how to split U into $U_1 \mid U_2$ by showing how to split each b_j into b'_i and b''_j such that:

$$\begin{array}{ll} b_j = b'_j + b''_j & (1) \\ a'_i \cdot n_i[T_i] \sim_{h,w,N} b'_j \cdot n_j[T_j] \quad \text{for } j \in 1..k & (2) \\ a''_i \cdot n_i[T_i] \sim_{h,w,N} b''_j \cdot n_j[T_j] \quad \text{for } j \in 1..k & (3) \\ a'_i \cdot T_i \sim_{h,w,N} b'_j \cdot T_j \quad \text{for } j \in k+1..s & (4) \\ a''_i \cdot T_i \sim_{h,w,N} b''_j \cdot T_j \quad \text{for } j \in k+1..s & (5) \end{array}$$

We choose these b'_j and b''_j exaclty as in [32], keeping into account $w = w_1 + w_2$ and then we define U (and U') as the tree obtained by parallel composition of b'_j components (and b''_j respectively). By repeated applications of Lemma 6.1.9 we have $T_1 \sim_{h,w_1,N} U_1$ and $T_2 \sim_{h,w_1,N} U_2$ respectively. □

Now we define a size $|A|^{h,w,N}$ for each formula A, and we prove that a formula with a size (h, w, N) cannot distinguish between trees $T \sim_{h,w,N} U$.

Table 6.1.1. *Size of logical formulas with names*

$\|\mathbf{F}\|^h$	$= 0$	$\|\mathbf{F}\|^w$	$= 0$	$\|\mathbf{F}\|^N$	$= \emptyset$
$\|A \wedge B\|^h$	$= max(\|A\|^h, \|B\|^h)$	$\|A \wedge B\|^w$	$= max(\|A\|^w, \|B\|^w)$	$\|A \wedge B\|^N$	$= \|A\|^N \cup \|B\|^N$
$\|A \Rightarrow B\|^h$	$= max(\|A\|^h, \|B\|^h)$	$\|A \Rightarrow B\|^w$	$= max(\|A\|^w, \|B\|^w)$	$\|A \Rightarrow B\|^N$	$= \|A\|^N \cup \|B\|^N$
$\|\mathbf{0}\|^h$	$= 1$	$\|\mathbf{0}\|^w$	$= 1$	$\|\mathbf{0}\|^N$	$= \emptyset$
$\|n[A]\|^h$	$= 1 + \|A\|^h$	$\|n[A]\|^w$	$= max(2, \|A\|^w)$	$\|n[A]\|^N$	$= \{n\} \cup \|A\|^N$
$\|A@n\|^h$	$= max(\|A\|^h - 1, 0)$	$\|A@n\|^w$	$= \|A\|^w$	$\|A@n\|^N$	$= \|A\|^N \cup \{n\}$
$\|A \mid B\|^h$	$= max(\|A\|^h, \|B\|^h)$	$\|A \mid B\|^w$	$= \|A\|^w + \|B\|^w$	$\|A \mid B\|^N$	$= \|A\|^N \cup \|B\|^N$
$\|A \triangleright B\|^h$	$= \|B\|^h$	$\|A \triangleright B\|^w$	$= \|B\|^w$	$\|A \triangleright B\|^N$	$= \|B\|^N$
$\|A \oslash n\|^h$	$= \|A\|^h$	$\|A \oslash n\|^w$	$= \|A\|^w$	$\|A \oslash n\|^N$	$= \|A\|^N \cup \{n\}$

$|A|^{h,w,N} \stackrel{def}{=} (|A|^h, |A|^w, |A|^N)$

Proposition 6.1.11 (Formula can distinguish up to the size)**.**

$$|A|^{h,w,N} = (h, w, N),\ T \models A,\ T \sim_{h,w,N} U \ \Rightarrow\ U \models A \qquad (1).$$

Proof. By induction on the structure of A.

Case F. Immediate.

Case $A \wedge B$**.** Follows easily by Lemma 6.1.7 and induction.

Case $A \Rightarrow B$**.** Let $|A|^{h,w,N} = (h_1, w_1, N_1)$ and $|B|^{h,w,N} = (h_2, w_2, N_2)$. We have $|A \Rightarrow B|^{h,w,N} = (max(h_1, h_2), max(w_1, w_2), N_1 \cup N_2)$ and we have to prove that if $U \models A$ then $U \models B$. Suppose $U \models A$, by Lemma 6.1.7 we have $U \sim_{h_1,w_1,N_1} T$, and by induction we have $T \models A$. Now since $T \models A \Rightarrow B$, we have $T \models B$, by Lemma 6.1.7 $T \sim_{h_2,w_2,N_2} U$ and by induction $U \models B$.

Case 0. $|\mathbf{0}|^{h,w,N} = (1, 1, \emptyset)$ Suppose $T \models \mathbf{0}$ and $T \sim_{1,1,\emptyset} U$. Then $T \equiv \mathbf{0}$. Since $T \sim_{1,1,\emptyset} U$, if $U \not\equiv \mathbf{0}$ then $T \not\equiv \mathbf{0}$. Hence it must be $U \equiv \mathbf{0}$ and thus $U \models \mathbf{0}$.

Case $n[A]$**.** Let $|A|^{h,w,N} = (h, w, n)$. We have

$$|n[A]|^{h,w,N} = (h+1, max(w,2), N \cup \{n\})$$

and $T \sim_{h+1,max(w,2),N\cup\{n\}} U$ and $T \models n[A]$. Then there exists T' such that $T \equiv n[T']$ and $T' \models A$. By $T \sim_{h+1,max(w,2),N\cup\{n\}} U$ and $T \equiv n[T']$ we deduce that there exists U_1 s.t. $U \equiv n[U_1]$ and $U_1 \sim_{h,max(w,2),N\cup\{n\}} T'$. By Lemma 6.1.7 $U_1 \sim_{h,w,N} T'$ and by induction $U_1 \models A$. This proves $U \models n[A]$.

Case $A@n$**.** Let $|A|^{h,w,N} = (h, w, n)$. We have

$$|A@n|^{h,w,N} = (max(h-1, 0), w, N \cup \{n\})$$

and $T \sim_{max(h-1,0),w,N\cup\{n\}} U$ and $T \models A@n$. Then $n[T] \models A$. If $h > 0$ then $n[T] \sim_{h,w,N\cup\{n\}} n[U]$ by Lemma 6.1.9 If $h = 0$ then it is easy to see that $n[T] \sim_{0,w,N\cup\{n\}} n[U]$. In both cases we can apply the induction by the Lemma 6.1.7 obtaining $n[U] \models A$. This proves $U \models n[A]$.

Case $A \mid B$**.** Let $|A|^{h,w,N} = (h_1, w_1, N_1)$ and $|B|^{h,w,N} = (h_2, w_2, N_2)$. We have $|A \mid B|^{h,w,N} = (max(h_1, h_2), w_1 + w_2, N_1 \cup N_2)$ and $T \models A \mid B$. Then there exist T_1 and T_2 such that $T \equiv T_1 \mid T_2$ and $T_1 \models A$ and $T_2 \models B$. We have $T_1 \mid T_2 \sim_{max(h_1,h_2),w_1+w_2,N_1\cup N_2} U$ and by Lemma 6.1.8 there exist U_j such that $U_j \sim_{max(h_1,h_2),w_j,N_1\cup N_2} T_j$ for $j = 1, 2$. By induction Lemma 6.1.7 and induction we have $U_1 \models A$ and $U_2 \models B$. This proves $U \models A \mid B$.

Case $A \triangleright B$**.** Let $|B|^{h,w,N} = (h, w, N)$ and $T \models A \triangleright B$. We have $|A \triangleright B|^{h,w,N} = (h, w, N)$ and $T \sim_{h,w,N} U$. Consider any T_1 such that $T_1 \models A$. Then $T_1 \mid T \models B$. Since $T_1 \sim_{h,w,N} T_1$ then by Lemma 6.1.9 $T_1 \mid T \sim_{h,w,N} T_1 \mid U$. By induction we have $T_1 \mid U \models B$. This proves $U \models A \triangleright B$.

Case $A \oslash n$**.** Let $|A|^{h,w,N} = (h, w, N)$, we have $|A \triangleright B|^{h,w,N} = (h, w, N \cup \{n\})$ and $T \sim_{h,w,N\cup\{n\}} U$. By Lemma 6.1.9 since $n \in N \cup \{n\}$ we have $(\nu n)\, T \sim_{h,w,N\cup\{n\}} (\nu n)\, U$ and by Lemma 6.1.7 $(\nu n)\, T \sim_{h,w,N} (\nu n)\, U$. By $T \models A \oslash n$ we have $(\nu n)\, T \models A$ and by induction $(\nu n)\, U \models A$. This proves $U \models A \oslash n$ □

6.1.2 Enumerating Equivalence Classes

Also here we extend the approach of [32]. We begin introducing some notation for describing equivalence classes.

Notation 6.1.12. *We use the metavariable c ranges over sets of trees modulo structural congruence and the following notation*

$$
\begin{array}{lcl}
\langle T\rangle_{\equiv} & \stackrel{def}{=} & \{T' \mid T \equiv T'\} \\
\langle T\rangle_{\sim_{h,w,N}} & \stackrel{def}{=} & \{T' \mid T \sim_{h,w,N} T'\} \\
\Sigma_{i\in S} T_i & \stackrel{def}{=} & \bigcup_{i\in S} T_i \\
n[c] & \stackrel{def}{=} & \{\langle n[T]\rangle_{\equiv} \mid \langle T\rangle_{\equiv} \in c\} \\
(\nu m)\, T & \stackrel{def}{=} & \{\langle (\nu m)\, T\rangle_{\equiv} \mid \langle T\rangle_{\equiv} \in c\} \\
c^{\leq w} & \stackrel{def}{=} & \{\langle a_1 \cdot T_1 \mid \ldots \mid a_k \cdot T_k\rangle_{\equiv} \quad \mid \quad 0 \leq a_i \leq w \text{ for } i \in 1..k\}
\end{array}
$$

We can now give a direct definition of the set of equivalence classes $NF(h, w, N)$ determined by $\sim_{h,w,N}$.

Definition 6.1.13. *Let N be a finite set of names, we define $NF(h, w, N)$ as follows:*

$$
\begin{array}{l}
NF(0, w, N) \stackrel{def}{=} (\Sigma_{M\subset N}\ (\nu m)\, m[M])^{\leq w} \\
NF(h+1, w, N) \stackrel{def}{=} (\Sigma_{n\in N}\ n[NF(h, w, N)] + \Sigma_{M\subset N}\ b_M \cdot (\nu m)\, m[M])^{\leq w}
\end{array}
$$

Lemma 6.1.14. *For each T and for each (h, w, N) there exists $U \sim_{h,w,N} T$ such that $U \in NF(h, w, N)$.*

Proof. We can construct a witness for each equivalence class by pruning sealed trees into the smallest sealed tree containing the same free names. After this we prune the subtrees exceeding the width w. □

From this it follows that $NF(h, w, N)$ enumerates the witnesses for each equivalence class of $\sim_{h,w,N}$.

Corollary 6.1.15 (Enumerating)**.** *For each (h, w, N) the set $NF(h, w, N)$ is a finite enumeration of (h, w, N) equivalence classes.*

6.1.3 Decidability on abstract trees

With the help of the previous propositions we can finally prove the desired result.

Theorem 6.1.16 (Adding Restricted Names and Revelation Adjunct)**.**
The model-checking problem restricted to closed formulas generated by the following grammar ($\exists, \mathsf{И}, \mathbf{H}$, Ⓡ: no, ⊘: yes):

$$A ::= \mathbf{F} \mid \mathbf{0} \mid A \Rightarrow A \mid n[A] \mid A \mid A \mid A \triangleright A \mid A@n \mid A \oslash n$$

is decidable over all trees (i.e., including trees with restricted names).

Proof. It is enough to find an algorithm to decide whether $T \models A \triangleright B$, model-checking the other operators is easy. Now $T \models A \triangleright B$ is by definition $\forall T' \in \mathcal{T}.T' \models A \Rightarrow T \mid T' \models B$.

If $|B|^{h,w,N} = (h, w, N)$, by Proposition 6.1.11 we can reduce the quantification on the infinite set of terms to a quantification on the set of equivalence classes witnesses of $\sim_{h,w,N}$. By Proposition 6.1.15 we can produce an enumeration $U_i^{(h,w,N)}$ of a witness for each equivalence class of $\sim_{h,w,N}$. Thus checking $T \models A \triangleright B$ can be reduced to checking that, for each $U_i^{(h,w,N)}$, $U_i^{(h,w,N)} \models A \Rightarrow U_i^{(h,w,N)} \mid T \models B$. □

We show now that $\textcircled{\scriptsize C}\, n$ can be encoded in terms of revelation adjunct $A \oslash n$.

Lemma 6.1.17 (Presence from Revelation Adjunct)**.** *Given $n \neq m$:*

$$\textcircled{\scriptsize C}\, n \dashv\vdash (n[\mathbf{0}] \triangleright ((\neg(\neg\mathbf{0} \mid \neg\mathbf{0})) \oslash n))@m$$

Proof.

$$\begin{array}{rcl} & T \models (n[\mathbf{0}] \triangleright ((\neg(\neg\mathbf{0} \mid \neg\mathbf{0})) \oslash n))@m \\ \Leftrightarrow & m[T] \models n[\mathbf{0}] \triangleright ((\neg(\neg\mathbf{0} \mid \neg\mathbf{0})) \oslash n) \\ \Leftrightarrow & m[T] \mid n[\mathbf{0}] \models (\neg(\neg\mathbf{0} \mid \neg\mathbf{0})) \oslash n \\ \Leftrightarrow & (\nu n)\,(m[T] \mid n[\mathbf{0}]) \models \neg(\neg\mathbf{0} \mid \neg\mathbf{0}) \\ (1) \quad \Leftrightarrow & \forall T_1, T_2.\, T_1 \mid T_2 \equiv (\nu n)\,(m[T] \mid n[\mathbf{0}]) \Rightarrow T_1 \equiv \mathbf{0} \vee T_2 \equiv \mathbf{0} \end{array}$$

Now if $n \in fn(T)$ then the tree $U = (\nu n)\,(m[T] \mid n[\mathbf{0}])$ is sealed, thus it can be splitted into $U' \mid T_{\mathbf{0}}$ and $T_{\mathbf{0}} \mid U'$ only (where $U' \equiv U$ and $T_{\mathbf{0}} \equiv \mathbf{0}$) and (1) is true. If $n \notin fn(T)$ then $(\nu n)\,(m[T] \mid n[\mathbf{0}]) \equiv m[T] \mid (\nu n)\, n[\mathbf{0}]$ and (1) is false. Hence (1) $\Leftrightarrow n \in fn(T) \Leftrightarrow T \models \textcircled{\scriptsize C}\, n$ □

Note that the previous encoding $\textcircled{\scriptsize c}^m \eta = (\eta[\mathbf{0}] \triangleright ((\neg(\neg\mathbf{0} \mid \neg\mathbf{0})) \oslash \eta))@m$ is not applicable when η is a variable, since the encoding relies on m never clashing with η. For example: $\mathbf{0} \not\models \exists x.\, \textcircled{\scriptsize C}\, x$ but $\mathbf{0} \models \exists x.\, \textcircled{\scriptsize c}^m x$ since $\mathbf{0} \models \textcircled{\scriptsize c}^m m$. However we can use Lemma 6.1.17 to encode the general case: given two names m, m' such that $m \neq m'$, $\textcircled{\scriptsize C}\, \eta \dashv\vdash \textcircled{\scriptsize c}^m \eta \wedge \textcircled{\scriptsize c}^{m'} \eta$.

$$\begin{array}{lcl} & & T, \rho \models \textcircled{\scriptsize c}^m \eta \wedge \textcircled{\scriptsize c}^{m'} \eta \Leftrightarrow \text{ (by cases)} \\ \eta\rho = m & \Rightarrow & T \models \mathbf{T} \wedge \textcircled{\scriptsize c}^{m'} m \Leftrightarrow T, \rho \models \textcircled{\scriptsize C}\, \eta \\ \eta\rho = m' & \Rightarrow & T \models \textcircled{\scriptsize c}^m m' \wedge \mathbf{T} \Leftrightarrow T, \rho \models \textcircled{\scriptsize C}\, \eta \\ \eta\rho \neq m, \eta\rho \neq m' & \Rightarrow & T \models \textcircled{\scriptsize c}^m \eta\rho \wedge T \models \textcircled{\scriptsize c}^{m'} \eta\rho \Leftrightarrow T, \rho \models \textcircled{\scriptsize C}\, \eta \end{array}$$

Of course, $И y.\, \textcircled{\scriptsize c}^y \eta$, where y is a fresh variable, would work as well, but we are trying to encode $\textcircled{\scriptsize C}\, \eta$ without quantifiers. The encoding gives us the following corollary.

Corollary 6.1.18 (Adding $\textcircled{\scriptsize c}$)**.** *The model-checking problem for closed formulas in $SL_{\{\oslash, @\}}$ is decidable over all trees (i.e., including trees with restricted names).*

6.2 Quantifier Extrusion

Table 6.2.1. *Extrusion of existential quantifier*

$x \notin fv(B)$	$(\forall x.\, A) \wedge B$	$\dashv\vdash$	$\forall x.\, (A \wedge B)$	$(\forall\text{-}\wedge)$	$(\exists x.\, A) \wedge B$	$\dashv\vdash$	$\exists x.\, (A \wedge B)$	$(\exists\text{-}\wedge)$
	$\neg(\forall x.\, A)$	$\dashv\vdash$	$\exists x.\, (\neg A)$	$(\forall\text{-}\neg)$	$\neg(\exists x.\, A)$	$\dashv\vdash$	$\forall x.\, (\neg A)$	$(\exists\text{-}\neg)$
$y \neq \eta$	$\eta[\forall y.\, A]$	$\dashv\vdash$	$\forall y.\, (\eta[A])$	$(\forall\text{-}[])$	$\eta[\exists y.\, A]$	$\dashv\vdash$	$\exists y.\, (\eta[A])$	$(\exists\text{-}[])$
$x \notin fv(B)$	$(\forall x.\, A) \mid B$	$\vdash$	$\forall x.\, (A \mid B)$	$(\forall\text{-}\mid\vdash)$	$(\exists x.\, A) \mid B$	$\dashv\vdash$	$\exists x.\, (A \mid B)$	$(\exists\text{-}\mid)$
$y \neq x$	$Иx.\, \forall y.\, A$	$\vdash$	$\forall y.\, (Иx.\, A)$	$(\forall\text{-}И\vdash)$	$Иx.\, \exists y.\, A$	$\dashv$	$\exists y.\, (Иx.\, A)$	$(\exists\text{-}И\dashv)$

	$m\textcircled{R}\forall y.\, A$	$\vdash$	$\forall y.\, (m\textcircled{R}A)$	$(\forall\text{-}\textcircled{R}\vdash)$	$m\textcircled{R}\exists y.\, A$	$\dashv\vdash$	$\exists y.\, (m\textcircled{R}A)$	$(\exists\text{-}\textcircled{R})$
$x \notin fv(B)$	$(\forall x.\, A) \triangleright B$	$\dashv$	$\exists x.\, (A \triangleright B)$	$(\forall\text{-}\triangleright l \dashv)$	$(\exists x.\, A) \triangleright B$	$\dashv\vdash$	$\forall x.\, (A \triangleright B)$	$(\exists\text{-}\triangleright\, l)$
$x \notin fv(A)$	$A \triangleright (\forall x.\, B)$	$\dashv\vdash$	$\forall x.\, (A \triangleright B)$	$(\forall\text{-}\triangleright\, r)$	$A \triangleright (\exists x.\, B)$	$\dashv$	$\exists x.\, (A \triangleright B)$	$(\exists\text{-}\triangleright r \dashv)$
$y \neq \eta$	$(\forall y.\, A)@\eta$	$\dashv\vdash$	$\forall y.\, (A@\eta)$	$(\forall\text{-}@)$	$(\exists y.\, A)@\eta$	$\dashv\vdash$	$\exists y.\, (A@\eta)$	$(\exists\text{-}@)$
$y \neq \eta$	$(\forall y.\, A)\oslash\eta$	$\dashv\vdash$	$\forall y.\, (A\oslash\eta)$	$(\forall\text{-}\oslash)$	$(\exists y.\, A)\oslash\eta$	$\dashv\vdash$	$\exists y.\, (A\oslash\eta)$	$(\exists\text{-}\oslash)$

We start our discussion of extrusion on a familiar ground, by listing, in Table 6.2.1, some logical equivalences that can be used to extrude universal and existential quantifiers from some of the other operators. The first four are the usual First Order Logic (FOL) rules.

If all the rules were double implications ($\dashv\vdash$), we could use them to extrude the existential quantifier in any formula, thanks to Lemma 5.2.4. However, the presence of some single implications prevents their direct use for this aim. Each simple implication we write is actually strict, i.e. whenever we write $A \vdash B$ in the table above we also mean that $B \vdash A$ has a counterexample. We prove this fact by exhibiting, for any such schematic implication, an instance $A' \vdash B'$ and a tree T such that $T \not\models A'$ and $T \models B'$. This proves that $B \vdash A$ cannot be valid. Below, we will often use a name m as an abbreviation for $m[\mathbf{0}]$. We write $T \models A$ when any tree satisfies A, and $T \not\models A$ when no tree satisfies A. The notation $\eta \in \{\eta_1, \ldots, \eta_i\}$ stands for the formula $\eta = \eta_1 \vee \ldots \vee \eta = \eta_i$.

$(\forall\text{-} \mid \not\vdash)$	$n[\mathbf{0}] \mid m[\mathbf{0}] \not\models (\forall x.\, x \in \{n, m\} \Rightarrow x[\mathbf{0}]) \mid \mathbf{T}$
	$n[\mathbf{0}] \mid m[\mathbf{0}] \models \forall x.\, (x \in \{n, m\} \Rightarrow x[\mathbf{0}] \mid \mathbf{T})$
$(\forall\text{-}\mathsf{И}\ \not\vdash)$	$T \not\models \mathsf{И}x.\, \forall y.\, x \neq y$
	$T \models \forall y.\, \mathsf{И}x.\, x \neq y$
$(\exists\text{-}\mathsf{И}\ \not\forall)$	$T \not\models \exists y.\, \mathsf{И}x.\, x = y$
	$T \models \mathsf{И}x.\, \exists y.\, x = y$
$(\forall\text{-}\textcircled{R}\ \not\vdash)$	$(\nu n)\,(\nu n')\, n[m] \mid n'[m'] \not\models p\textcircled{R}\forall x.\, (p[\neg\, \textcircled{C}\, x] \mid \mathbf{T})$
	$(\nu n)\,(\nu n')\, n[m] \mid n'[m'] \models \forall x.\, p\textcircled{R}(p[\neg\, \textcircled{C}\, x] \mid \mathbf{T})$
$(\forall\text{-}\triangleright l\ \not\vdash)$	$T \not\models \exists x.\, (x[\mathbf{0}] \triangleright \mathbf{F})$
	$T \models (\forall x.\, x[\mathbf{0}]) \triangleright \mathbf{F}$
$(\exists\text{-}\triangleright r\ \not\forall)$	$T \not\models \exists x.\, (\mathbf{T} \triangleright \neg\, \textcircled{C}\, x)$
	$T \models \mathbf{T} \triangleright (\exists x.\, \neg\, \textcircled{C}\, x)$

The table above shows that $\forall$-$\exists$ extrusion is not trivial, but it does not prove it to be impossible (for example, simple double-implication rules for $\exists$-$\mathsf{И}$ and $\forall$-$\mathsf{И}$ do exist); the actual impossibility proof will come later. Similar rules, riddled with single implications, govern the extrusion of hiding quantifiers and of $\textcircled{R}$. In this case as well, we will show later that they cannot be adjusted.

Table 6.2.2. *Extrusion of freshness quantifier*

$x \notin fv(B)$	$(И x.\, A) \wedge B$	$\dashv\vdash$	$И x.\, (A \wedge B)$	$(И\text{-}\wedge)$
	$\neg(И x.\, A)$	$\dashv\vdash$	$И x.\, (\neg A)$	$(И\text{-}\neg)$
$y \neq \eta$	$\eta[И y.\, A]$	$\dashv\vdash$	$И y.\, (\eta[A])$	$(И\text{-}[])$
$x \notin fv(B)$	$(И x.\, A) \mid B$	$\dashv\vdash$	$И x.\, (A \mid B)$	$(И\text{-}\mid)$
$y \neq x$	$\exists x.\, И y.\, A$	$\vdash$	$И y.\, (\exists x.\, A)$	$(И\text{-}\exists\ \vdash)$
$y \neq \eta$	$\eta Ⓡ И y.\, A$	$\dashv\vdash$	$И y.\, (\eta Ⓡ A)$	$(И\text{-}Ⓡ)$
$x \notin fv(B)$	$(И x.\, A) \triangleright B$	$\dashv$	$И x.\, (A \triangleright B)$	$(И\text{-}\triangleright l\ \dashv)$
$x \notin fv(A)$	$A \triangleright (И x.\, B)$	$\dashv$	$И x.\, (A \triangleright B)$	$(И\text{-}\triangleright r\ \dashv)$
$y \neq \eta$	$(И y.\, A)@\eta$	$\dashv\vdash$	$И y.\, (A@\eta)$	$(И\text{-}@)$
$y \neq \eta$	$(И y.\, A)\oslash\eta$	$\dashv\vdash$	$И y.\, (A\oslash\eta)$	$(И\text{-}\oslash)$

The situation looks very similar for the freshness quantifier (Table 6.2.6.2), apart from the fact that, thanks to its self-duality, we only need half of the rules.
Once more, all the single implications are strict.

$(И\text{-}\exists\ \not\dashv)$
$$(\nu n)\, n[\mathbf{0}] \not\models \exists x.\, И y.\, y Ⓡ Ⓒ x$$
$$(\nu n)\, n[\mathbf{0}] \models И y.\, \exists x.\, y Ⓡ Ⓒ x$$

$(И\text{-}\triangleright l\ \not\vdash)$
$$\begin{aligned}
&T \not\models И x.\, (Ⓒ x \triangleright \mathbf{F}) \\
&\Leftrightarrow \neg(\forall n.\ n \notin fn(T) \Rightarrow T \models Ⓒ n \triangleright \mathbf{F}) \\
&\Leftrightarrow \neg(\forall n.\ n \notin fn(T) \Rightarrow \forall U.\ U \models Ⓒ n \Rightarrow T \mid U \models \mathbf{F}) \\
&\Leftrightarrow \neg(\forall n.\ n \notin fn(T) \Rightarrow \forall U.\ U \models \neg Ⓒ n) \\
&\Leftrightarrow \exists n.\ n \notin fn(T) \wedge \exists U.\ U \models Ⓒ n \\
&\text{consider } U = n[\mathbf{0}]
\end{aligned}$$
$$\begin{aligned}
&T \models (И x.\, Ⓒ x) \triangleright \mathbf{F} \\
&\Leftrightarrow \forall U.\ U \models И x.\, Ⓒ x \Rightarrow T \mid U \models \mathbf{F} \\
&\Leftrightarrow \forall U.\ U \not\models И x.\, Ⓒ x \\
&\Leftrightarrow \forall U, n.\ n \notin fn(U) \Rightarrow U \not\models Ⓒ n \\
&\Leftrightarrow \forall U, n.\ n \notin fn(U) \Rightarrow U \models \neg Ⓒ n
\end{aligned}$$

$(И\text{-}\triangleright r\ \not\vdash)$
$$\begin{aligned}
&T \not\models И x.\, (\mathbf{T} \triangleright \neg Ⓒ x) \\
&\Leftrightarrow \neg(\forall n.\ n \notin fn(T) \Rightarrow T \models \mathbf{T} \triangleright \neg Ⓒ n) \\
&\Leftrightarrow \neg(\forall n, U.\ n \notin fn(T) \Rightarrow T \mid U \models \neg Ⓒ n) \\
&\Leftrightarrow \exists n, U.\ n \notin fn(T) \wedge T \mid U \models Ⓒ n \\
&\text{consider } U = n[\mathbf{0}]
\end{aligned}$$
$$\begin{aligned}
&T \models \mathbf{T} \triangleright (И x.\, \neg Ⓒ x) \\
&\Leftrightarrow \forall U.\ T \mid U \models И x.\, \neg Ⓒ x \\
&\Leftrightarrow \forall U, n.\ n \notin fn(T, U) \Rightarrow T \mid U \models \neg Ⓒ n
\end{aligned}$$

However, the three single-implication rules admit a double-implication version, as shown in the Table 6.2.6.2.

Table 6.2.3. *Extrusion of freshness quantifier - part two*

$x \neq y$	$\exists x.\, И y.\, A$	$\dashv\vdash$	$И y.\, (\exists x.\, A \wedge x \neq y)$	$(И\text{-}\exists)$
$y \notin fv(B)$	$(И y.\, A) \triangleright B$	$\dashv\vdash$	$И y.\, ((\neg \copyright y \wedge A) \triangleright B)$	$(И\text{-} \triangleright l)$
$y \notin fv(A)$	$A \triangleright (И y.\, B)$	$\dashv\vdash$	$И y.\, ((\neg \copyright y \wedge A) \triangleright B)$	$(И\text{-} \triangleright r)$

The last two rules are bizarre: regardless of which side (of $\triangleright$) $И$ is extruded from, y must always be excluded from the left hand side. The next Lemma shows that this is indeed the case.

Lemma 6.2.1 (Extrusion of freshness)**.** *There is an algorithm to transform any formula in the full logic into an equivalent formula in $И$-prenex form.*

Proof. The algorithm exhaustively applies the double-implication rules of Tables 6.2.6.2 and 6.2.6.2, left to right, until possible. The result is equivalent to the original formula thanks to Lemma 5.2.4. Termination is easy.

To prove the correctness of the rules, we must prove that any ground instance of the left hand side is equivalent to the corresponding instance of the right hand side. We assume that ρ is an arbitrary ground substitution defined on all the free variables of the involved formulas (in all the rules both sides have the same free variables). We will also assume that all bound variables in A (and in B) are different, and that ρ is not defined on those variables (hence, in all cases below we assume that ρ is not defined on either x or y).

In the proof we will make extensive use of Corollary 5.2.12, which expresses the fundamental semantic property of the Gabbay-Pitts freshness quantifier.
$(И\text{-}\wedge)$, $(И\text{-}\neg)$, $(И\text{-}\ |)$: see [44].

$(И\text{-}[])$: assume $y \neq \eta$, hence $y \neq (\eta\rho)$.

$T \models (\eta[И y.\, A])\rho \Leftrightarrow$
$T \models \eta\rho[И y.\, A\rho] \Leftrightarrow$
$\exists T'.\, T \equiv \eta\rho[T'] \ \wedge\ T' \models И y.\, A\rho \Leftrightarrow$
$\exists T'.\, T \equiv \eta\rho[T'] \ \wedge\ \exists m \notin fn(T', A\rho).\, T' \models A\rho\{y \leftarrow m\} \Leftrightarrow$
By Corollary 5.2.12
$\exists T'.\, T \equiv \eta\rho[T'] \ \wedge\ \exists m \notin (fn(T', A\rho) \cup \{\eta\rho\}).\, T' \models A\rho\{y \leftarrow m\} \Leftrightarrow$
By Lemma 5.1.2, we have that $fn(T', A\rho) \cup \{\eta\rho\} = fn(T, \eta\rho, A\rho)$
$\exists T'.\, T \equiv \eta\rho[T'] \ \wedge\ \exists m \notin fn(T, \eta\rho, A\rho).\, T' \models A\rho\{y \leftarrow m\} \Leftrightarrow$
$\exists m \notin fn(T, \eta\rho, A\rho).\, \exists T'.\, T \equiv \eta\rho[T'] \ \wedge\ T' \models A\rho\{y \leftarrow m\} \Leftrightarrow$
$\exists m \notin fn(T, \eta\rho, A\rho).\, T \models \eta\rho[A\rho\{y \leftarrow m\}] \Leftrightarrow$ (By $y \neq \eta\rho$)
$\exists m \notin fn(T, \eta\rho, A\rho).\, T \models (\eta\rho[A\rho])\{y \leftarrow m\} \Leftrightarrow$
$T \models И y.\, \eta\rho[A\rho] \Leftrightarrow$
$T \models (И y.\, \eta[A])\rho$

($\mathsf{И}$-Ⓡ) with $y \neq \eta\rho$
$T \models (\eta \text{Ⓡ} \mathsf{И} y.\, A)\rho \Leftrightarrow$
$T \models \eta\rho \text{Ⓡ} \mathsf{И} y.\, A\rho \Leftrightarrow$
$\exists T'.\, T \equiv (\nu\eta\rho)\, T' \ \wedge\ T' \models \mathsf{И} y.\, A\rho \Leftrightarrow$
$\exists T'.\, T \equiv (\nu\eta\rho)\, T' \ \wedge\ \exists m \notin fn(T', A\rho).\, T' \models A\rho\{y \leftarrow m\} \Leftrightarrow$
By Corollary 5.2.12
$\exists T'.\, T \equiv (\nu\eta\rho)\, T' \ \wedge\ \exists m \notin (fn(T', A\rho) \cup \{\eta\rho\}).\, T' \models A\rho\{y \leftarrow m\} \Leftrightarrow$
By Lemma 5.1.2, we have that $fn(T') \cup \{\eta\rho\} = fn(T, \eta\rho)$
$\exists T'.\, T \equiv (\nu\eta\rho)\, T' \ \wedge\ \exists m \notin fn(T, \eta\rho, A\rho).\, T' \models A\rho\{y \leftarrow m\} \Leftrightarrow$
$\exists m \notin fn(T, \eta\rho, A\rho).\, \exists T'.\, T \equiv (\nu\eta\rho)\, T' \ \wedge\ T' \models A\rho\{y \leftarrow m\} \Leftrightarrow$
$\exists m \notin fn(T, \eta\rho, A\rho).\, T \models \eta\rho \text{Ⓡ} A\rho\{y \leftarrow m\} \Leftrightarrow$ (By $y \neq \eta\rho$)
$\exists m \notin fn(T, \eta\rho, A\rho).\, T \models (\eta\rho \text{Ⓡ} A\rho)\{y \leftarrow m\} \Leftrightarrow$
$T \models \mathsf{И} y.\, \eta\rho \text{Ⓡ} A\rho \Leftrightarrow$
$T \models (\mathsf{И} y.\, \eta \text{Ⓡ} A)\rho$

($\mathsf{И}$-@) with $y \neq \eta\rho$
$T \models (\mathsf{И} y.\, A@\eta)\rho \Leftrightarrow$
$T \models \mathsf{И} y.\, A\rho@\eta\rho \Leftrightarrow$
$\eta\rho[T] \models \mathsf{И} y.\, A\rho \Leftrightarrow$
$\exists m \notin fn(\eta\rho[T], A\rho).\, \eta\rho[T] \models A\rho\{y \leftarrow m\} \Leftrightarrow$
$\exists m \notin fn(T, \eta\rho, A\rho).\, T \models A\rho\{y \leftarrow m\}@\eta\rho \Leftrightarrow$ (By $y \neq \eta\rho$)
$\exists m \notin fn(T, \eta\rho, A\rho).\, T \models (A\rho@\eta\rho)\{y \leftarrow m\} \Leftrightarrow$
$T \models \mathsf{И} y.\, A\rho@\eta\rho \Leftrightarrow$
$T \models (\mathsf{И} y.\, A@\eta)\rho$

($\mathsf{И}$-⦸) with $y \neq \eta\rho$
$T \models ((\mathsf{И} y.\, A) \oslash \eta)\rho \Leftrightarrow$
$T \models (\mathsf{И} y.\, A\rho) \oslash \eta\rho \Leftrightarrow$
$(\nu\eta\rho)\, T \models (\mathsf{И} y.\, A\rho) \Leftrightarrow$
$\exists m \notin fn((\nu\eta\rho)\, T, A\rho).\, (\nu\eta\rho)\, T \models A\rho\{y \leftarrow m\} \Leftrightarrow$
$\exists m \notin (fn((\nu\eta\rho)\, T, A\rho) \cup \{\eta\rho\}).\, (\nu\eta\rho)\, T \models A\rho\{y \leftarrow m\} \Leftrightarrow$
$\exists m \notin fn(T, \eta\rho, A\rho).\, T \models A\rho\{y \leftarrow m\} \oslash \eta\rho \Leftrightarrow$ (By $y \neq \eta\rho$)
$\exists m \notin fn(T, \eta\rho, A\rho).\, T \models (A\rho \oslash \eta\rho)\{y \leftarrow m\} \Leftrightarrow$
$T \models \mathsf{И} y.\, A\rho \oslash \eta\rho \Leftrightarrow$
$T \models (\mathsf{И} y.\, A \oslash \eta)\rho$

($\mathsf{И}$-$\exists$) Assume $x \neq y$. In the following proof, we use A^y_x, A^m_x, A^y_n, and A^m_n, to abbreviate $A\rho$, $A\rho\{y \leftarrow m\}$, $A\rho\{x \leftarrow n\}$, and $A\rho\{x \leftarrow n\}\{y \leftarrow m\}$, respectively. $A\rho\{x \leftarrow n\}\{y \leftarrow m\}$ and $A\rho\{y \leftarrow m\}\{x \leftarrow n\}$ are equal by $x \neq y$, hence are both abbreviated as A^m_n.

$T \models (\exists x.\, \text{И} y.\, A)\rho \Leftrightarrow$
$T \models \exists x.\, \text{И} y.\, A^y_x \Leftrightarrow$
$\exists n.\ T \models \text{И} y.\, A^y_n \Leftrightarrow$
$\exists n.\ \exists m \notin fn(T, A^y_n).\ T \models A^m_n \Leftrightarrow$ (Corollary 5.2.12)
$\exists n.\ \exists m \notin (fn(T, A^y_x) \cup \{n\}).\ T \models A^m_n \Leftrightarrow$
$\exists m \notin fn(T, A^y_x).\ \exists n.\ m \neq n \wedge T \models A^m_n \Leftrightarrow$
$\exists m \notin fn(T, A^y_x).\ \exists n.\ T \models (A^m_n \wedge m \neq n) \Leftrightarrow$
$\exists m \notin fn(T, A^y_x).\ T \models \exists x.\, (A^m_x \wedge m \neq x) \Leftrightarrow$
$\exists m \notin fn(T, A^y_x).\ T \models (\exists x.\, (A^y_x \wedge y \neq x))\{y \leftarrow m\} \Leftrightarrow$
$\exists m \notin fn(T, \exists x.\, (A^y_x \wedge y \neq x)).\ T \models (\exists x.\, (A^y_x \wedge y \neq x))\{y \leftarrow m\} \Leftrightarrow$
$T \models \text{И} y.\, (\exists x.\, A\rho \wedge x \neq y) \Leftrightarrow$
$T \models (\text{И} y.\, (\exists x.\, A \wedge x \neq y))\rho$

($\text{И}\text{-} \triangleright l$) Assume $y \notin fv(B)$.

$T \models ((\text{И} y.\, A) \triangleright B)\rho \Leftrightarrow$
$T \models (\text{И} y.\, A\rho) \triangleright B\rho \Leftrightarrow$
$\forall T'.\ T' \models \text{И} y.\, A\rho \ \Rightarrow\ T \mid T' \models B\rho \Leftrightarrow$
$\forall T'.\ (\exists n \notin fn(T', A\rho).\ T' \models A\rho\{y \leftarrow n\}) \ \Rightarrow\ T \mid T' \models B\rho \Leftrightarrow$
$\forall T'.\ (\exists n \notin fn(T, T', A\rho, B\rho).\ T' \models A\rho\{y \leftarrow n\}) \ \Rightarrow\ T \mid T' \models B\rho \Leftrightarrow$
$\forall T'.\ \forall n \notin fn(T, T', A\rho, B\rho).\ (T' \models A\rho\{y \leftarrow n\} \ \Rightarrow\ T \mid T' \models B\rho) \Leftrightarrow$
$\forall n \notin fn(T, A\rho, B\rho).$
$\qquad \forall T'.\ n \notin fn(T') \Rightarrow (T' \models A\rho\{y \leftarrow n\}) \ \Rightarrow\ T \mid T' \models B\rho) \Leftrightarrow$
$\forall n \notin fn(T, A\rho, B\rho).$
$\qquad \forall T'.\ (n \notin fn(T') \ \wedge\ T' \models A\rho\{y \leftarrow n\}) \ \Rightarrow\ T \mid T' \models B\rho \Leftrightarrow$
$\forall n \notin fn(T, A\rho, B\rho).\ \forall T'.\ (T' \models \neg\, \textcircled{\scriptsize C}\, n \wedge A\rho\{y \leftarrow n\}) \ \Rightarrow\ T \mid T' \models B\rho \Leftrightarrow$
$\forall n \notin fn(T, A\rho, B\rho).\ \forall T'.\ T' \models (\neg\, \textcircled{\scriptsize C}\, y \wedge A\rho)\{y \leftarrow n\} \ \Rightarrow\ T \mid T' \models B\rho \Leftrightarrow$
$\forall n \notin fn(T, A\rho, B\rho).\ T \models (\neg\, \textcircled{\scriptsize C}\, y \wedge A\rho)\{y \leftarrow n\} \triangleright B\rho \Leftrightarrow$
$\forall n \notin fn(T, A\rho, B\rho).\ T \models ((\neg\, \textcircled{\scriptsize C}\, y \wedge A\rho) \triangleright B\rho)\{y \leftarrow n\} \Leftrightarrow$
$T \models \text{И} y.\, ((\neg\, \textcircled{\scriptsize C}\, y \wedge A\rho) \triangleright B\rho)$
$T \models (\text{И} y.\, (\neg\, \textcircled{\scriptsize C}\, y \wedge A) \triangleright B)\rho$

($\text{И}\text{-} \triangleright r$) Assume $y \notin fv(A)$.

$T \models (A \triangleright \text{И} y.\, B)\rho \Leftrightarrow$
$T \models A\rho \triangleright \text{И} y.\, B\rho \Leftrightarrow$
$\forall T'.\ T' \models A\rho \ \Rightarrow\ T \mid T' \models \text{И} y.\, B\rho \Leftrightarrow$
$\forall T'.\ T' \models A\rho \ \Rightarrow\ (\forall n \notin fn(T, T', B\rho).\ T \mid T' \models B\rho\{y \leftarrow n\}) \Leftrightarrow$ (Cor. 5.2.12)
$\forall T'.\ T' \models A\rho \ \Rightarrow\ (\forall n \notin fn(T, T', A\rho, B\rho).\ T \mid T' \models B\rho\{y \leftarrow n\}) \Leftrightarrow$
$\forall T'.\ \forall n \notin fn(T, T', A\rho, B\rho).\ (T' \models A\rho \ \Rightarrow\ T \mid T' \models B\rho\{y \leftarrow n\}) \Leftrightarrow$
$\forall n \notin fn(T, A\rho, B\rho).\ \forall T'.\ n \notin fn(T') \Rightarrow (T' \models A\rho \ \Rightarrow\ T \mid T' \models B\rho\{y \leftarrow n\}) \Leftrightarrow$

$\forall n \notin fn(T, A\rho, B\rho).\ \forall T'.\ (n \notin fn(T') \wedge T' \models A\rho) \ \Rightarrow\ T \mid T' \models B\rho\{y \leftarrow n\} \Leftrightarrow$
$\forall n \notin fn(T, A\rho, B\rho).\ \forall T'.\ (T' \models \neg\, \textcircled{\scriptsize C}\, n \wedge A\rho) \ \Rightarrow\ T \mid T' \models B\rho\{y \leftarrow n\} \Leftrightarrow$
$\forall n \notin fn(T, A\rho, B\rho).\ T \models (\neg\, \textcircled{\scriptsize C}\, n \wedge A\rho) \triangleright (B\rho\{y \leftarrow n\}) \Leftrightarrow$ (By $y \notin fv(A)$)
$\forall n \notin fn(T, A\rho, B\rho).\ T \models (\neg\, \textcircled{\scriptsize C}\, y \wedge A\rho \triangleright B\rho)\{y \leftarrow n\} \Leftrightarrow$
$T \models \mathsf{И} y.\ ((\neg\, \textcircled{\scriptsize C}\, y \wedge A\rho) \triangleright B\rho)$
$T \models (\mathsf{И} y.\ (\neg\, \textcircled{\scriptsize C}\, y \wedge A) \triangleright B)\rho$

□

We now use this result to prove decidability of the freshness quantifier.

6.3 Decidability and Extrusion Results

We first observe that *model-checking* is decidable for prenex logics; of course, this is not true, in general, for validity, or for model-checking non-prenex formulas.

Theorem 6.3.1 (Decidability of Prenex Model-Checking)**.** *Model-checking over all trees is decidable for the closed formulas F generated by the following grammar ($\exists$, $\mathbf{H}$, $\textcircled{\scriptsize R}$, $\mathsf{И}$: outermost only; $\textcircled{\scriptsize c}$, $\oslash$: unlimited):*

$$F ::= \exists x.\, F \mid x\,\textcircled{\scriptsize R}\,F \mid \mathbf{H}x.\, F \mid \mathsf{И}x.\, F \mid \neg F \mid A$$
$$A ::= \mathbf{0} \mid \eta[A] \mid A \mid A \mid A \wedge A \mid \neg A \mid \ \textcircled{\scriptsize C}\,\eta \mid A \triangleright A \mid A@\eta \mid A \oslash \eta$$

Proof. By induction on the size of F and by cases.

Case $\neg F$ is trivial induction.

Case A is Corollary 6.1.18.

To model-check $T \models \exists x.\, F$, consider a finite set $\mathbf{N}$ of names containing $fn(T, F)$ plus one more fresh name m and model-check $T \models F\{x \leftarrow n\}$ for $n \in \mathbf{N}$. No other name needs to be considered, by Lemma 5.2.10.

To model-check $T \models n\,\textcircled{\scriptsize R}\,F$, transform T in ENF and apply Lemma 5.2.5.

To model-check $T \models \mathbf{H}x.\, F$, transform T in ENF and apply Lemma 5.2.7.

To model-check $T \models \mathsf{И}x.\, F$, choose a name $n \notin fn(T, F)$ and model-check $T \models F\{x \leftarrow n\}$. The result does not depend on the name by Corollary 5.2.12. □

Corollary 6.3.2 (Extrusion implies Decidability)**.** *The existence of an extrusion algorithm, i.e. an algorithm that transforms every formula into an equivalent formula generated by the grammar of Theorem 6.3.1, for any sublogic L of $SL_{\{\exists, \mathsf{И}, \oslash, \mathsf{H}, \textcircled{\scriptsize R}\}}$ containing $\triangleright$ implies the decidability of L.*

Proof. To decide $\mathbf{vld}(A)$ for a closed formula A, reduce it to $\mathbf{0} \models \mathbf{T} \triangleright A$, apply the extrusion algorithm, and use the algorithm of Theorem 6.3.1. □

As a consequence, the addition of freshness preserves the decidability of the logic of Corollary 6.1.18.

Corollary 6.3.3 (Decidability of Fresh Quantifiers)**.** *Model-checking and validity for the closed formulas in $SL_{\{\mathsf{И},\mathbin{\diamondsuit},\copyright\}}$ are decidable over all trees.*

Proof. Model-checking: we apply the algorithm of Lemma 6.2.1 to transform the formula in И-prenex form. This can be model-checked by Theorem 6.3.1.

Validity: given a formula A, we extrude the freshness quantifier from $\mathbf{T} \triangleright A$, obtaining B. We then model-check $\mathbf{0} \models \vec{\forall} B$, which is decidable by Theorem 6.3.1. By the property in Table 5.2.1, line 3, $\mathbf{0} \models \vec{\forall} B$ iff A is valid. □

To sum up, fresh quantification alone is not enough to lose decidability, even if combined with a limited form of revelation ($\copyright\,\eta$).

The proof is based on the possibility of extruding freshness quantifiers through all operators, including negation and the parallel adjunct operator that internalizes validity in the logic. This reveals a deep algebraic difference between freshness and existential quantification, where such extrusion is not possible. We now formalize this fact.

By Fact 7.2.1, Corollary 7.4.1 and Corollary 7.4.2, the three logics $SL_{\{\exists\}}$, $SL_{\{\textcircled{R}\}}$, and $SL_{\{\mathsf{H}\}}$ are all undecidable. Hence, we have the following Corollary.

Corollary 6.3.4 (No Extrusion)**.** *No extrusion algorithm (as defined in Corollary 6.3.2) exists for the existential quantifier, the revelation operator, and the hiding quantifier.*

Chapter 7

Undecidability of Revelation and Hiding

7.1 Standard Model

In this section we focus on a tiny sublogic of SL that contains the revelation operator and show that for each formula A of that sublogic, when a tree T satisfies A, there exists a cut-down version of T that satisfies the same formula. This is a key technical tool in order to prove (later) that the decidability of this tiny logic is already as hard as decidability of first order logic.

Notation 7.1.1 (Path-Formulas)**.** *A path-formula p is a formula denoting the existence of a path of edges, starting from the root and leading to a leaf, as follows (we only define path formulas of length one and two, since we need no more).*

$$.\eta \stackrel{def}{=} \eta[\mathbf{0}] \mid \mathbf{T} \qquad\qquad .\eta'.\eta \stackrel{def}{=} \eta'[\eta[\mathbf{0}] \mid \mathbf{T}] \mid \mathbf{T}$$

When a tree satisfies $.m.n$ we say that it "contains a path $m.n$"; the path ends with a leaf. The minimal tree containing such path, $m[n[\mathbf{0}]]$ (which we also write $m[n]$), is called a "line for the path $m.n$", and similarly $m[\mathbf{0}]$ (abbreviated as m) is a line for m.

We now introduce a notion of path cutting. Intuitively, the tree $Cut_N(T)$ contains one line for each of those paths $m.n$ of T such that m and n are either bound or in N (longer paths, and paths with free names not in N, are cut away). By this construction, for any formula A with shape $.n_1.n_2$, $n_1\textcircled{R}.n_2.n_3$, $n_1\textcircled{R}n_2\textcircled{R}.n_3.n_4$ (where n_i may be equal to n_j), $Cut_{nm(A)}(T)$ is A-equivalent to T, i.e. $Cut_{nm(A)}(T) \models A$ iff $T \models A$. Moreover, $Cut_N(T)$ contains a list $n_1[\mathbf{0}] \mid \ldots \mid n_j[\mathbf{0}]$, where $\{n_i\}^{i\in\{1..j\}} = fn(T) \cap N$, so that the validity of formulas $n\textcircled{R}\mathbf{T}$, for $n \in N$, is preserved as well. In other words, we cut away long paths and paths with free names not in N, and we rewrite trees like "$n[m \mid p]$" as lines "$n[m] \mid n[p] \mid n \mid m \mid p$".

We will prove that this cut-down structure is logically equivalent to the original tree, with respect to those formulas that only contain path-formulas of length 2 and

names that are in N (Corollary 7.1.16).

Before giving the formal definition, we give some examples. Cutting is only defined up-to-congruence.

flattening	$Cut_{\{n,m\}}(n[m \mid n])$	$\equiv$	$n[m]\|n[n] \mid n\|m$
cutting long paths	$Cut_{\{n,m\}}(n[m[n]])$	$\equiv$	$n\|m$
cutting w.r.t. more names	$Cut_{\{n,m,p\}}(n[m \mid n])$	$\equiv$	$n[m]\|n[n] \mid n\|m$
deleting free names	$Cut_{\{n\}}(n[m \mid n])$	$\equiv$	$n[n] \mid n$
preserving bound names	$Cut_{\{n\}}((\nu m)\, n[m \mid n])$	$\equiv$	$(\nu m)\, n[m]\|n[n] \mid n\|m$
name clashes don't matter	$Cut_{\{n,m\}}((\nu m)\, n[m \mid n])$	$\equiv$	$(\nu m)\, n[m]\|n[n] \mid n\|m$
preserving the name m	$Cut_{\{n,m\}}(n[n] \mid m[p])$	$\equiv$	$n[n] \mid n\|m$

We first define an auxiliary partial function $enfCut_N(T)$, that is only defined on trees in ENF, and is deterministic (while $Cut_N(T)$ is total but is defined only up to congruence). $enfCut_N(T)$ behaves as $Cut_N(T)$ in all the examples above. Then we define $Cut_N(T)$ by closing $enfCut_N(T)$ with respect to tree equivalence.

Definition 7.1.2 (Path cutting for ENF)**.** *For each tree in ENF, for each set of names N, we define the operation $enfCut_N()$ as follows. $Par\{T : cond\}$ combines (using $|$) all instances $(T)\sigma$ of T such that $(cond)\sigma$ is satisfied.*

$$enfCut_N((\nu m)\, T) \stackrel{def}{=} (\nu m)\, enfCut_{N\cup\{m\}}(U)$$

$$enfCut_N(U) \quad \text{(where } U \text{ contains no } (\nu n)\, A' \text{ subterm)} \stackrel{def}{=} Par\{n_1[n_2[\mathbf{0}]] : U \models .n_1.n_2,\ \{n_1, n_2\} \subseteq N\} \ \mid\ Par\{n[\mathbf{0}] : n \in (fn(U) \cap N)\}$$

Remark 7.1.3 (Cutting, Reordering, and Renaming)**.** *Some remarks about cutting:*

1. *In definition 7.1.2 we do not require that $\{m_j\}$ and N are disjoint. This is exploited in Corollary 7.1.9.*

2. *$enfCut_N((\vec{\nu}_{i\in\{1,\ldots,j\}} n_i)\, U)$, i.e. $(\vec{\nu}_{i\in\{1,\ldots,j\}} n_i)\, enfCut_{N\cup\{n_1,\ldots,n_j\}}(U)$, is always in ENF: all the names in $\{n_i\}^{i\in\{1,\ldots,j\}}$ are free in U, hence they appear in an edge $n_i[\mathbf{0}]$ of the result.*

3. *$fn(enfCut_N(T)) = fn(T) \cap N$*

4. *If T' has the same matrix of T and a reordered prefix, then $enfCut_N(T')$ has the same matrix as $enfCut_N(T)$ (modulo the edge order, which is not fixed) but has the prefix of T'. In other terms, when the prefix of the input of $enfCut_N()$ is reordered, the prefix of the output is reordered in the same way.*

Lemma 7.1.4 (Congruence of $enfCut_N()$)**.** *if T and T' are in ENF then*

$$T \equiv T' \;\Rightarrow\; enfCut_N(T) \equiv enfCut_N(T')$$

Proof. We consider the case when $N \subseteq fn(T)$, hence, by congruence, $N \subseteq fn(T')$. The general case follows immediately by observing that $enfCut_N(T) = enfCut_{N \cap fn(T)}(T)$.

By Lemma 5.1.7, $T \equiv T'$ implies that exist $U, T'', U'', U', \{n_i\}^{i\in\{1..j\}}, \{n'_i\}^{i\in\{1..j\}}$, and a bijection τ, such that all the U's are restriction-free and

$$\begin{array}{lcl} T & = & (\vec{\nu}_{i\in\{1,\ldots,j\}} n_i)\, U \\ T'' & = & (\vec{\nu}_{i\in\{1,\ldots,j\}} n_i)\, U'' \\ T' & = & (\vec{\nu}_{i\in\{1,\ldots,j\}} n'_i)\, U' \end{array}$$

with

- $N \# \{n_i\}^{i\in\{1..j\}}$ and $N \# \{n'_i\}^{i\in\{1..j\}}$, by $N \subseteq fn(T) = fn(T')$;
- $U \equiv U''$;
- $U'' = U'\{n'_l \leftarrow \tau(n_l)\}^{l\in\{1..j\}}$.

$U \equiv U''$ implies that $enfCut_N(U) \equiv enfCut_N(U'')$, since $fn(U) = fn(U'')$ and $U \models .n_1.n_2$ iff $U'' \models .n_1.n_2$, hence $enfCut_N(T) \equiv enfCut_N(T'')$.

Similarly, $U'' = U'\{n'_l \leftarrow \tau(n_l)\}^{l\in\{1..j\}}$. implies that

$$\begin{array}{l} enfCut_{N\cup\{n_i\}^{i\in\{1..j\}}}(U'') \\ = enfCut_{N\cup\{n_i\}^{i\in\{1..j\}}}(U'\{n'_l \leftarrow \tau(n_l)\}^{l\in\{1..j\}}) \\ = (enfCut_{N\cup\{n'_i\}^{i\in\{1..j\}}}(U'))\{n'_l \leftarrow \tau(n_l)\}^{l\in\{1..j\}} \end{array}$$

hence

$$\begin{array}{l} enfCut_N(T'') = (\vec{\nu}_{i\in\{1,\ldots,j\}} n_i)\; enfCut_{N\cup\{n_i\}^{i\in\{1..j\}}}(U'') \\ = (\vec{\nu}_{i\in\{1,\ldots,j\}} n_i)\,(enfCut_{N\cup\{n'_i\}^{i\in\{1..j\}}}(U')\{n'_l \leftarrow \tau(n_l)\}^{l\in\{1..j\}}) \\ \equiv (\vec{\nu}_{i\in\{1,\ldots,j\}} n'_i)\; enfCut_{N\cup\{n'_i\}^{i\in\{1..j\}}}(U') \\ = enfCut_N((\vec{\nu}_{i\in\{1,\ldots,j\}} n'_i)\, U') \\ = enfCut_N(T') \end{array}$$

□

We now extend cutting from ENF to general terms.

Definition 7.1.5.

$$Cut_N(T) \;\triangleq\; \{enfCut_N(U) : U \in ENF(T)\}$$

Corollary 7.1.6 (Congruence of $Cut_N()$)**.**

$$T \equiv T' \;\wedge\; U \in Cut_N(T) \;\wedge\; U' \in Cut_N(T') \;\Rightarrow\; U \equiv U'$$

Proof. By definition,

$$\begin{array}{l} U \in \mathit{Cut}_N(T) \ \wedge \ U' \in \mathit{Cut}_N(T') \ \Rightarrow \\ \exists \bar{T}, \bar{T}'. \quad \bar{T} \in \mathit{ENF}(T), \ \mathit{enfCut}_N(\bar{T}) = U \\ \qquad\qquad \bar{T}' \in \mathit{ENF}(T'), \ \mathit{enfCut}_N(\bar{T}') = U' \end{array}$$

By transitivity, $\bar{T} \equiv \bar{T}'$. By Lemma 7.1.4, $U \equiv U'$. □

Because of the property above, hereafter we will always write, with a slight notational abuse, $T' = \mathit{Cut}_N(T)$ or $\mathit{Cut}_N(T) = T'$, instead of $T' \in \mathit{Cut}_N(T)$, i.e. we will have $\mathit{Cut}_N(T)$ standing for an arbitrary element of the set, whenever we are only interested in its value modulo congruence.

Lemma 7.1.7.

$$\mathit{fn}(\mathit{Cut}_N(T)) = \mathit{fn}(T) \cap N$$

Lemma 7.1.8.

$$\mathit{Cut}_N((\nu n)\, T) \equiv (\nu n)\, \mathit{Cut}_{N \cup \{n\}}(T)$$

Proof. If $n \notin \mathit{fn}(T)$, then it neither appears free in $\mathit{Cut}_{N\cup\{n\}}(T)$, hence the property holds trivially:

$$\mathit{Cut}_N((\nu n)\, T) \equiv \mathit{Cut}_N(T) \equiv \mathit{Cut}_{N\cup\{n\}}(T) \equiv (\nu n)\, \mathit{Cut}_{N\cup\{n\}}(T).$$

Otherwise, a ENF of $(\nu n)\, T$ is $(\nu n)\, T'$, where T' is a ENF of T, and

$$\begin{array}{l} \mathit{Cut}_N((\nu n)\, T) \equiv \mathit{enfCut}_N((\nu n)\, T') \equiv (\nu n)\, \mathit{enfCut}_{N\cup\{n\}}(T') \\ \equiv (\nu n)\, \mathit{Cut}_{N\cup\{n\}}(T). \end{array}$$

□

Corollary 7.1.9.

$$n \in N \ \Rightarrow \ \mathit{Cut}_N((\nu n)\, T) \equiv (\nu n)\, \mathit{Cut}_N(T)$$

Lemma 7.1.10 (Inversion)**.** *For any $n \notin \mathit{fn}(T)$:*

$$\mathit{Cut}_N(T) \equiv (\nu n)\, U' \Rightarrow \exists T'.\ T \equiv (\nu n)\, T' \wedge U' \equiv \mathit{Cut}_{N\cup\{n\}}(T')$$

Proof. If $n \notin \mathit{fn}(U')$ the thesis follows immediately with $T' = T$:

$$T \equiv (\nu n)\, T \ \wedge \ U' \equiv (\nu n)\, U' \equiv \mathit{Cut}_N(T) = \mathit{Cut}_{N\cup\{n\}}(T)$$

We consider now the case $n \in \mathit{fn}(U')$.
$\mathit{Cut}_N(T) \equiv (\nu n)\, U'$ means that

$$\exists \bar{T}, \bar{U}.\ \bar{T} \in \mathit{ENF}, \ \bar{U} \in \mathit{ENF}, \ T \equiv \bar{T}, \ \mathit{enfCut}_N(\bar{T}) = \bar{U}, \ \bar{U} \equiv (\nu n)\, U'$$

Choose a $\bar{\bar{U}} \in \mathit{ENF}$ with $\bar{\bar{U}} \equiv U'$. $\bar{U}$ and $(\nu n)\, \bar{\bar{U}}$ are congruent and ENF, hence they only differ for prefix reordering and renaming, and have equivalent matrixes. By $\bar{U} = \mathit{enfCut}_N(\bar{T})$, if we apply the same reordering and renaming that transform $\bar{U}$ into $(\nu n)\, \bar{\bar{U}}$ to $\bar{T}$, we get $\bar{\bar{T}} \equiv \bar{T}$ such that $\mathit{enfCut}_N(\bar{\bar{T}}) = (\nu n)\, \bar{\bar{U}}$. Hence, $\exists T'.\ \bar{\bar{T}} = (\nu n)\, T'$ and $\mathit{enfCut}_N(T') = \bar{\bar{U}}$. This is the thesis, since $\bar{\bar{U}} \equiv U'$. □

Corollary 7.1.11 (Inversion with $n \in N$)**.** *If $n \in N$, then:*

$$Cut_N(T) \equiv (\nu n)\, U' \Rightarrow \exists T'.\ T \equiv (\nu n)\, T' \wedge U' \equiv Cut_N(T')$$

Proof. $(\nu n)\, U' \in Cut_N(T)$ implies that n is not free in $Cut_N(T)$, hence, by Lemma 7.1.7 and $n \in N$, $n \notin fn(T)$. Then we apply the lemma above. □

Before proving our key lemma, we introduce the De Morgan dual of revelation. This is useful so that we can distribute negation down to the leaves of any formula in our logic.

Notation 7.1.12 (Corevelation)**.**

$$\eta \text{Ⓤ} A \quad \stackrel{def}{=} \quad \neg(\eta \text{Ⓡ} \neg A)$$

Lemma 7.1.13.

$$T \models n \text{Ⓤ} A \quad \Leftrightarrow \quad \forall T'.\ T \equiv (\nu n)\, T' \Rightarrow T' \models A$$

Lemma 7.1.14 (Standard Model)**.** *Let A be a closed formula generated by the following grammar:*

$$A ::= .\eta_1.\eta_2 \mid \neg.\eta_1.\eta_2 \mid A \wedge A \mid A \vee A \mid \eta \text{Ⓡ} A \mid \eta \text{Ⓤ} A \mid \mathsf{M}x.\, A$$

then: $T \models A \Rightarrow Cut_{nm(A)}(T) \models A$.

Proof. We prove the following stronger property, that goes better through induction, by induction on the size of A, and by cases:

$$\forall \mathbf{N} \text{ finite}.\ T \models A \Rightarrow Cut_{nm(A) \cup \mathbf{N}}(T) \models A.$$

If A is a path-formula $.n_1.n_2$ then $n_1[n_2[\mathbf{0}]]$ is in $Cut_{\{n_1,n_2\} \cup \mathbf{N}}(T)$ iff $T \models .n_1.n_2$, by construction. This proves the lemmas for both cases $.n_1.n_2$ and $\neg.n_1.n_2$. Observe that, by the closure hypothesis, we do not consider cases $.x.n$, $.n.x$, $.x.y$.

If $A = A' \wedge A''$, or $A = A' \vee A''$, the thesis follows by induction.

If $A = n \text{Ⓡ} A'$, then:

$$\begin{array}{ll}
T \models n \text{Ⓡ} A' & \Leftrightarrow \text{ def of } \text{Ⓡ} \\
\exists T'.\ T \equiv (\nu n)\, T',\ T' \models A' & \Rightarrow \text{ by ind.} \\
\exists T'.\ T \equiv (\nu n)\, T',\ \forall \mathbf{M}.\ Cut_{nm(A') \cup \mathbf{M}}(T') \models A' & \Rightarrow \mathbf{M} \leftarrow \mathbf{N} \cup \{n\} \\
\exists T'.\ T \equiv (\nu n)\, T',\ \forall \mathbf{N}.\ Cut_{nm(A') \cup \mathbf{N} \cup \{n\}}(T') \models A' & \Rightarrow \text{ def of } \text{Ⓡ} \\
\exists T'.\ T \equiv (\nu n)\, T',\ \forall \mathbf{N}.\ (\nu n)\, Cut_{nm(A') \cup \mathbf{N} \cup \{n\}}(T') \models n \text{Ⓡ} A' & \Leftrightarrow \text{ by Cor. 7.1.9} \\
\exists T'.\ T \equiv (\nu n)\, T',\ \forall \mathbf{N}.\ Cut_{nm(A') \cup \mathbf{N} \cup \{n\}}((\nu n)\, T') \models n \text{Ⓡ} A' & \Rightarrow \\
\forall \mathbf{N}.\ \exists T'.\ T \equiv (\nu n)\, T',\ Cut_{nm(A') \cup \mathbf{N} \cup \{n\}}((\nu n)\, T') \models n \text{Ⓡ} A' & \Leftrightarrow \text{ by } T \equiv (\nu n)\, T' \\
\forall \mathbf{N}.\ Cut_{nm(A') \cup \mathbf{N} \cup \{n\}}(T) \models n \text{Ⓡ} A' & \Leftrightarrow \\
\forall \mathbf{N}.\ Cut_{nm(n \text{Ⓡ} A') \cup \mathbf{N}}(T) \models n \text{Ⓡ} A' &
\end{array}$$

Now, assume $A = n\textcircled{U}A'$ and $T \models A$. We want to prove that.:

$$\begin{array}{ll} \forall \mathbf{N}.\ Cut_{nm(n\textcircled{U}A')\cup\mathbf{N}}(T) \models n\textcircled{U}A' & i.e. \\ \forall \mathbf{N}, T'.\ (\nu n)\, T' \equiv Cut_{nm(n\textcircled{U}A')\cup\mathbf{N}}(T) \Rightarrow T' \models A' & (1) \end{array}$$

We assumed that $T \models n\textcircled{U}A'$ i.e. $\forall T''.\ T \equiv (\nu n)\, T'' \Rightarrow T'' \models A'$; by induction:

$$\forall T'', \mathbf{M}.\ T \equiv (\nu n)\, T'' \Rightarrow Cut_{nm(A')\cup\mathbf{M}}(T'') \models A' \quad (2)$$

To prove (1), we assume $(\nu n)\, T' \equiv Cut_{nm(n\textcircled{U}A')\cup\mathbf{N}}(T)$ (a).
Since $n \in nm(n\textcircled{U}A') \cup \mathbf{N}$, we can apply Corollary 7.1.11 to (a), obtaining that:

$$\begin{array}{ll} \exists T'''. & T \equiv (\nu n)\, T''' \\ & \wedge\ T' \equiv Cut_{nm(n\textcircled{U}A')\cup\mathbf{N}}(T''') = Cut_{nm(A')\cup\mathbf{N}\cup\{n\}}(T''') \quad (3) \end{array}$$

Hence we can apply (2), with $T'' \leftarrow T'''$ and $\mathbf{M} \leftarrow \mathbf{N} \cup \{n\}$, obtaining

$$Cut_{nm(A')\cup\mathbf{N}\cup\{n\}}(T''') \models A',$$

which implies the thesis $T' \models A'$, since, by (3)

$$T' \equiv Cut_{nm(A')\cup\mathbf{N}\cup\{n\}}(T''').$$

If $A = \mathsf{И}x.\, A'$, then:

$$\begin{array}{ll} T \models \mathsf{И}x.\, A' & \Leftrightarrow \text{def of } \mathsf{И} \\ \forall \mathbf{N}.\ \exists n.\ n \notin (fn(T, A') \cup \mathbf{N}) & \\ \qquad \wedge\ T \models A'\{x \leftarrow n\} & \Rightarrow \text{by ind.} \\ \forall \mathbf{N}.\ \exists n.\ n \notin (fn(T, A') \cup \mathbf{N}) & \\ \qquad \wedge\ \forall \mathbf{M}.\ Cut_{nm(A'\{x \leftarrow n\})\cup\mathbf{M}}(T) \models A'\{x \leftarrow n\} & \Rightarrow \\ \forall \mathbf{N}.\ \exists n.\ n \notin (fn(T, A') \cup \mathbf{N}) & \\ \qquad \wedge\ Cut_{nm(A'\{x \leftarrow n\})\cup\mathbf{N}}(T) \models A'\{x \leftarrow n\} & \Leftrightarrow \text{by } n \notin fn(T) \\ \forall \mathbf{N}.\ \exists n.\ n \notin (fn(T, A') \cup \mathbf{N}) & \\ \qquad \wedge\ Cut_{nm(A'\{x \leftarrow n\})\setminus\{n\}\cup\mathbf{N}}(T) \models A'\{x \leftarrow n\} & \Leftrightarrow \text{by } n \notin nm(A') \\ \forall \mathbf{N}.\ \exists n.\ n \notin (fn(T, A') \cup \mathbf{N}) & \\ \qquad \wedge\ Cut_{nm(\mathsf{И}x.\, A')\cup\mathbf{N}}(T) \models A'\{x \leftarrow n\} & \Leftrightarrow \text{by Cor. 5.2.12 } (3 \Leftrightarrow 5) \\ \forall \mathbf{N}.\ Cut_{nm(\mathsf{И}x.\, A')\cup\mathbf{N}}(T) \models \mathsf{И}x.\, A' & \end{array}$$

□

Corollary 7.1.15 (Standard Model)**.** *Let A be a closed formula generated by the following grammar:*

$$A ::= .\eta_1.\eta_2 \mid A \wedge A \mid \eta\textcircled{R}A \mid \mathsf{И}x.\, A \mid \neg A$$

then: $T \models A \Rightarrow Cut_{nm(A)}(T) \models A$.

Corollary 7.1.16 (Standard Model)**.** *Let A be a closed formula generated by the following grammar:*

$$A ::= .\eta_1.\eta_2 \mid A \wedge A \mid \eta \textcircled{R} A \mid \mathsf{M}x.\, A \mid \neg A$$

then: $T \models A \Leftrightarrow Cut_{nm(A)}(T) \models A$.

Proof. $T \models A \Rightarrow Cut_{nm(A)}(T) \models A$ by Corollary 7.1.15.

For the other direction, assume $T \not\models A$; then:

$$\begin{array}{ll} T \not\models A & \Leftrightarrow \text{def of } \neg A \\ T \models \neg A & \Rightarrow \text{Corollary 7.1.15} \\ Cut_{nm(A)}(T) \models \neg A & \Leftrightarrow \text{def of } \neg A \\ Cut_{nm(A)}(T) \not\models A & \end{array}$$

□

7.2 Encoding Revelation in FOL

Since we are studying undecidability, we focus here on weak versions of the logic. We will prove undecidability for a logic with just $\wedge$, $\neg$, $\textcircled{R}$, and path formulas. The undecidability of any richer logic follows immediately.

A known undecidability result for spatial logic is the following.

Fact 7.2.1 (Undecidability of Existential Quantification)**.** *Validity of closed formulas built from $\exists x.\, A$, $A \wedge A$, $\neg A$, $x[A] \mid \mathbf{T}$ is not decidable.*

Proof. Proved in [50], by encoding any first-order formula whose vocabulary is just a binary relation in the fragment above. The undecidability over finite trees follows by Trakhtenbrot Theorem. The undecidability over infinite trees follows by Church-Turing Theorem. □

We are going to define a translation of FOL formulas into SL formulas, and FOL structures into SL trees, in order to reduce SL satisfiability to FOL satisfiability over a finite domain, which is known to be undecidable.

We first define our specific flavour of FOL. We consider formulas over a vocabulary which only consists of a binary relation R, i.e. formulas generated by the following grammar (this logic is already undecidable [16]):

$$\phi \ ::= \ \exists x.\, \phi \mid \phi \wedge \psi \mid \neg\phi \mid R(x, x')$$

We define satisfaction of a closed formula, over an interpretation consisting of a domain $\mathcal{D}$ and a binary relation $\mathcal{R}$ over $\mathcal{D}$, with respect to a variable assignment σ

with $\sigma{\downarrow} \supseteq fv(\phi)$ (where $f{\downarrow}$ is the domain of a function f) as follows.

$$\begin{array}{lll}
\mathcal{D},\mathcal{R},\sigma \models \exists x.\,\phi & \Leftrightarrow_{\text{def}} & \text{exists } c{\in}\mathcal{D}.\ \mathcal{D},\mathcal{R},\sigma\{x{\leftarrow}c\} \models \phi \\
\mathcal{D},\mathcal{R},\sigma \models \phi \wedge \psi & \Leftrightarrow_{\text{def}} & \mathcal{D},\mathcal{R},\sigma \models \phi \ \text{ and } \ \mathcal{D},\mathcal{R},\sigma \models \psi \\
\mathcal{D},\mathcal{R},\sigma \models \neg\phi & \Leftrightarrow_{\text{def}} & \text{not } (\mathcal{D},\mathcal{R},\sigma \models \phi) \\
\mathcal{D},\mathcal{R},\sigma \models R(x,x') & \Leftrightarrow_{\text{def}} & (\sigma(x),\sigma(x')) \in \mathcal{R}
\end{array}$$

Essentially, we will translate a model $\mathcal{D},\mathcal{R}$ into an ENF term $(\vec{\nu}n_i)\,[\![\mathcal{D}]\!] \mid [\![\mathcal{R}]\!]$, with one name n_i for each element of $\mathcal{D}$, with $\mathcal{R}$ encoded as set of lines of length two, and $\mathcal{D}$ encoded as a set of lines of length one, obtaining structures that have the same shape as the cut-down trees introduced in Section 7.1.

In the formula, we will translate $\exists$ into $\textcircled{\text{R}}$ and $R(x,y)$ into $.m.n$. To translate $\exists$ into $\textcircled{\text{R}}$, we have to overcome some differences between the two operators. The most important difference is the fact that $\exists$ is a binder while $\textcircled{\text{R}}$ is not. In FOL semantics, we associate each variable x that is bound in a formula $\exists x.\phi$ with a value c that is "free" in the domain. In the SL translation this becomes an association between a name m that is free in a formula $m\textcircled{\text{R}}A$ and a name n_i that is *bound* in the model $(\vec{\nu}n_i)\,T$. So, while in FOL we match variables in the formula with values in the domain, in the SL translation we will match bound names in the model with the free names used to reveal them in the formula.

Technically, we translate a FOL closed formula ϕ into a formula $[\![\phi]\!]$, where all the closed variables of ϕ are left open, and a ground substitution $(\!|\phi|\!)^{\mathbf{P}}$ such that $(\!|\phi|\!)^{\mathbf{P}}{\downarrow} \supseteq fv(\phi)$, so that $[\![\phi]\!](\!|\phi|\!)^{\mathbf{P}}$ is closed. We then reduce satisfiability of ϕ to satisfiability of (a variant of) $[\![\phi]\!](\!|\phi|\!)^{\mathbf{P}}$.

A second difference is the fact that the same value can be bound to two different FOL variables, while the same restricted name cannot be revealed twice, hence, $\{(c,c)\} \models \exists x_1.\ \exists x_2.\ R(x_1,x_2)$ but $(\nu n)\,n[n[\mathbf{0}]] \not\models n_1\textcircled{\text{R}}n_2\textcircled{\text{R}}.n_1.n_2$.

We solve this problem by translating $\exists x_1.\ \exists x_2.\ \phi$ as if it were

$$\exists x_1.\ ((\exists x_2 \neq x_1.\ \phi) \vee \phi\{x_2{\leftarrow}x_1\}),\ \text{i.e. as: } x_1\textcircled{\text{R}}((x_2\textcircled{\text{R}}[\![\phi]\!]) \vee [\![\phi\{x_2{\leftarrow}x_1\}]\!]),$$

To this aim, in the translation algorithm a parameter $\mathbf{Y}$ keeps track of the quantified variables met during the translation. The first line of Table 7.2.7.2.7 defines how $\mathbf{Y}$ is grown with each quantification, and how it is used to generate a disjunction of $[\![\phi\{x_2{\leftarrow}x_1\}]\!]^{\mathbf{Y}}$ clauses.

Finally, while x in $\exists x.\ \phi$ can only be associated to an element that is in the domain, n in $n\textcircled{\text{R}}A$ can also be associated to a name that does not appear in the model at all (see Lemma 5.2.5). We solve this problem by translating $\exists x.\,\phi$ as $x\textcircled{\text{R}}([\![\phi]\!] \wedge .x)$ and by restricting our attention to models where, for every name n in a term, a line $n[\mathbf{0}]$ is present. We use our results on tree-cutting to show that this restriction is without loss of generality.

Notation 7.2.2 (Disjointness)**.**

$$K \# K' \quad \Leftrightarrow_{def} \quad K \cap K' = \emptyset$$

Notation 7.2.3. *We write* $M : \mathbf{M} \overset{in}{\rightharpoonup} \mathbf{N}$ *to specify that* M *is partial and injective from* $\mathbf{M}$ *to* $\mathbf{N}$*, and* $M : \mathbf{M} \overset{in}{\to} \mathbf{N}$ *to specify that* M *is total and injective from* $\mathbf{M}$ *to* $\mathbf{N}$*. For any partial function* $N : \mathbf{M} \rightharpoonup \mathbf{N}$*, we will use* $N{\downarrow}$ *to denote its actual domain and* $N{\uparrow}$ *to denote its actual range, i.e.:*

$$N{\downarrow} = \{m : \exists n \in \mathbf{N}.\ N(m) = n\} \quad N{\uparrow} = \{n : \exists m \in \mathbf{M}.\ N(m) = n\}$$

Notation 7.2.4. $(\vec{\nu}_{i\in I} n_i)\, T \triangleq (\nu n_{i_1}) \ldots (\nu n_{i_j})\, T$ *with* $I = \{i_1, \ldots, i_j\}$, $n : I \overset{in}{\to} \Lambda$.

Notation 7.2.5 (Bound Variables)**.** *We use* $bv(\phi)$ *to denote the set of all the variables bound in* ϕ*. We will always assume that all bound variables in a formula are distinct.*

Notation 7.2.6. *When* $M, N : \mathbf{M} \rightharpoonup \mathbf{N}$*, we use* $M \oplus N$ *to denote function extension, as follows:* $M \oplus N(x) \triangleq$ *if* $x \in N{\downarrow}$ *then* $N(x)$ *else* $M(x)$
Hence, $M \oplus \{c \leftarrow n\}$ *yields* n *on* c *and coincides with* M *elsewhere.*

$M \setminus c$ *is undefined on* c *and coincides with* M *elsewhere.*

When ρ *and* ρ' *are two substitutions, we define* $\rho; \rho'$ *as the only substitution such that:* $A(\rho; \rho') = (A\rho)\rho'$*, and* $(\rho; \rho')(x) = \rho'(\rho(x))$ *(e.g.,* $\oplus\{x \leftarrow y\}; \oplus\{y \leftarrow c\} = \oplus\{x \leftarrow c\} \oplus \{y \leftarrow c\}$*.) Hence,* $\rho; \rho' = \rho\rho'$ *for any pair of ground substitutions.*

We can finally define our translation. We map an FOL formula to an SL formula, an interpretation $\mathcal{D}, \mathcal{R}$ to a tree $[\![\mathcal{D}, \mathcal{R}]\!]^{M,N}$, and a variable assignment to a ground substitution. The translation is parametrized on a couple of functions, M and N, with disjoint domains and ranges, such that $M \oplus N$ (see Notation 7.2.6) injectively maps the whole $\mathcal{D}$ into Λ. In a nutshell, elements in $M{\downarrow}$ are mapped into names that are free in $[\![\mathcal{D}, \mathcal{R}]\!]^{M,N}$, while $N{\downarrow}$ is mapped over bound names.

Definition 7.2.7 (Formula translation)**.** *We define here a translation of FOL formulas, interpretations, and variable assignments, into SL formulas, interpretations, and variable assignments. Moreover, each FOL formula* ϕ *is also mapped to a ground substitution, defined on all and only the bound variables in* ϕ*, which we assume to be mutually distinct. The translation is parametric with respect to a subset* $\mathbf{P}$ *of* Λ*, and to a couple of functions* M*,* N *such that* $M \oplus N : \mathcal{D} \overset{in}{\to} \Lambda$*.* $\mathbf{P}$ *is used to express freshness as "not belonging to* $\mathbf{P}$*". In the first clause of the "formulas into substitutions" we do not specify how* m' *is chosen, but we will assume that the choice is deterministic, i.e. that* $(\!|\phi|\!)^{\mathbf{P}}$ *is uniquely determined.*

Table 7.2.1. Formula translation

formulas into formulas		
$[\![\exists x.\, \phi]\!]^{\mathbf{Y}}$	$\overset{def}{=}$	$x \mathbin{\textcircled{R}} ([\![\phi]\!]^{\mathbf{Y} \cup \{x\}} \wedge .x) \vee \bigvee_{y \in \mathbf{Y}} [\![\phi\{x \leftarrow y\}]\!]^{\mathbf{Y}}$
$[\![\phi \wedge \psi]\!]^{\mathbf{Y}}$	$\overset{def}{=}$	$[\![\phi]\!]^{\mathbf{Y}} \wedge [\![\psi]\!]^{\mathbf{Y}}$
$[\![\neg\phi]\!]^{\mathbf{Y}}$	$\overset{def}{=}$	$\neg [\![\phi]\!]^{\mathbf{Y}}$
$[\![R(x, x')]\!]^{\mathbf{Y}}$	$\overset{def}{=}$	$.x.x'$

formulas into substitutions

$$
\begin{array}{lcl}
(\!|\exists x.\,\phi|\!)^{\mathbf{P}} & \stackrel{def}{=} & (\!|\phi|\!)^{\mathbf{P}} \oplus \{x \leftarrow m'\} \quad \textit{choose } m' \in \Lambda \setminus (\mathbf{P} \cup (\!|\phi|\!)^{\mathbf{P}}\uparrow) \\
(\!|\phi \wedge \psi|\!)^{\mathbf{P}} & \stackrel{def}{=} & (\!|\phi|\!)^{\mathbf{P}} \oplus (\!|\psi|\!)^{\mathbf{P} \cup (\!|\phi|\!)^{\mathbf{P}}\uparrow} \\
(\!|\neg\phi|\!)^{\mathbf{P}} & \stackrel{def}{=} & (\!|\phi|\!)^{\mathbf{P}} \\
(\!|R(x,x')|\!)^{\mathbf{P}} & \stackrel{def}{=} & \emptyset
\end{array}
$$

interpretations, domains, and relations into trees

$$
\begin{array}{lcl}
[\![\mathcal{D},\mathcal{R}]\!]^{M,N} & \stackrel{def}{=} & (\vec{\nu}_{c \in N\downarrow} N(c))\,([\![\mathcal{D}]\!]^{M \oplus N} \mid [\![\mathcal{R}]\!]^{M \oplus N}) \\
[\![\emptyset]\!]^{M} & \stackrel{def}{=} & \mathbf{0} \\
[\![\{c\} \cup \mathcal{D}]\!]^{M} & \stackrel{def}{=} & M(c)[\mathbf{0}] \mid [\![\mathcal{D}]\!]^{M} \\
[\![\{(c,c')\} \cup \mathcal{R}]\!]^{M} & \stackrel{def}{=} & M(c)[M(c')[\mathbf{0}]] \mid [\![\mathcal{R}]\!]^{M}
\end{array}
$$

assignments into assignments

$$
\begin{array}{lcl}
[\![\sigma \oplus \{x \leftarrow c\}]\!]^{M} & \stackrel{def}{=} & [\![\sigma]\!]^{M} \oplus \{x \leftarrow M(c)\} \\
[\![\emptyset]\!]^{M} & \stackrel{def}{=} & \emptyset
\end{array}
$$

Lemma 7.2.8.

$$
\begin{array}{lcll}
 & & fn([\![\mathcal{D},\mathcal{R}]\!]^{M,N}) \subseteq M\uparrow & (0) \\
c \in M\downarrow \ \wedge\ M : M\downarrow \stackrel{in}{\rightharpoonup} \Lambda & \Rightarrow & [\![\mathcal{D}]\!]^{M[c \mapsto m]} = [\![\mathcal{D}]\!]^{M}\{M(c) \leftarrow m\} & (1) \\
c \in M\downarrow \ \wedge\ M : M\downarrow \stackrel{in}{\rightharpoonup} \Lambda & \Rightarrow & [\![\mathcal{R}]\!]^{M[c \mapsto m]} = [\![\mathcal{R}]\!]^{M}\{M(c) \leftarrow m\} & (2) \\
 & & [\![\sigma \oplus \{x \leftarrow c\}]\!]^{M} = [\![\sigma]\!]^{M}[x \mapsto M(c)] & (3) \\
c \notin \sigma\uparrow & \Rightarrow & [\![\sigma \oplus \{x \leftarrow c\}]\!]^{M[c \mapsto m]} = [\![\sigma]\!]^{M} \oplus \{x \leftarrow m\} & (4) \\
\{x,y\} \# bv(\phi),\ x \notin \mathbf{Y} & \Rightarrow & [\![\phi\{x \leftarrow y\}]\!]^{\mathbf{Y}} = [\![\phi]\!]^{\mathbf{Y}}\{x \leftarrow y\} & (5) \\
x \notin \sigma\downarrow,\ y \in \sigma\downarrow & \Rightarrow & [x \mapsto y]; [\![\sigma]\!]^{M} = [\![\sigma]\!]^{M} \oplus \{x \leftarrow M(\sigma(y))\} & (6) \\
x \notin \sigma\downarrow & \Rightarrow & \oplus\{x \leftarrow m\}[\![\sigma]\!]^{M} = [\![\sigma]\!]^{M} \oplus \{x \leftarrow m\} & (7)
\end{array}
$$

Proof.

(4)

$$
\begin{array}{lcll}
[\![\sigma \oplus \{x \leftarrow c\}]\!]^{M[c \mapsto m]} & = & [\![\sigma]\!]^{M[c \mapsto m]} \oplus \{x \leftarrow (M[c \mapsto m])(c)\} & \\
= \ [\![\sigma]\!]^{M[c \mapsto m]} \oplus \{x \leftarrow m\} & = & (\text{by } c \notin \sigma\uparrow) \quad [\![\sigma]\!]^{M} \oplus \{x \leftarrow m\} &
\end{array}
$$

(5) By induction and by cases. Case $\phi = \exists z.\,\psi$:

$$
\begin{array}{ll}
[\![(\exists z.\,\psi)\{x \leftarrow y\}]\!]^{\mathbf{Y}} & \text{by } x,y \neq z \\
= [\![\exists z.\,\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}} & \text{by def.} \\
= z\textcircled{R}([\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y} \cup \{z\}} \wedge .z) \vee \bigvee_{w \in \mathbf{Y}} [\![\psi\{x \leftarrow y\}\{z \leftarrow w\}]\!]^{\mathbf{Y}} & \text{by } x,y \neq z, \\
 & w \neq x \\
= z\textcircled{R}([\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y} \cup \{z\}} \wedge .z) \vee \bigvee_{w \in \mathbf{Y}} [\![\psi\{z \leftarrow w\}\{x \leftarrow y\}]\!]^{\mathbf{Y}} & \text{ind.} \\
 & (x \notin \mathbf{Y} \cup \{z\}) \\
= z\textcircled{R}([\![\psi]\!]^{\mathbf{Y} \cup \{z\}}\{x \leftarrow y\} \wedge .z) \vee \bigvee_{w \in \mathbf{Y}} [\![\psi\{z \leftarrow w\}]\!]^{\mathbf{Y}}\{x \leftarrow y\} & \text{by } x \neq z \\
= (z\textcircled{R}([\![\psi]\!]^{\mathbf{Y} \cup \{z\}} \wedge .z) \vee \bigvee_{w \in \mathbf{Y}} [\![\psi\{z \leftarrow w\}]\!]^{\mathbf{Y}})\{x \leftarrow y\} & \text{by def.} \\
= [\![\exists z.\,\psi]\!]^{\mathbf{Y}}\{x \leftarrow y\} &
\end{array}
$$

□

Theorem 7.2.9 (Faithfulness of translation)**.** *For any FOL formula ϕ, for any interpretation $(\mathcal{D},\mathcal{R})$, for any set of variables $\mathbf{Y}$, for any variable assignment σ, for any pair of partial injective functions $M,N : \mathcal{D} \stackrel{in}{\rightharpoonup} \Lambda$, for any finite set of names $\mathbf{P}$, such that:*

(a)	$\sigma\downarrow \supseteq fv(\phi) \cup \mathbf{Y}$	*σ closes the free variables of ϕ and those in $\mathbf{Y}$*
(b)	$\sigma\uparrow \subseteq M\downarrow$	*σ sends everything into the M-elements*
(c)	$\sigma(\mathbf{Y}) = M\downarrow$	*every M-element is reached by σ*
(d)	$M\downarrow \cup N\downarrow = \mathcal{D}$	*we know how to translate any $c \in \mathcal{D}$*
(e)	$M\downarrow \# N\downarrow$	*M-elements and N-elements are distinct*
(f)	$M\uparrow \# N\uparrow$	*M-names and N-names are distinct*
(g)	$\mathbf{P} \supseteq (M \oplus N)\uparrow$	*names not in $\mathbf{P}$ are "fresh"*
(h)	$bv(\phi) \# \sigma\downarrow$	*$bv(\phi) \# fv(\phi)$ and $bv(\phi) \# \mathbf{Y}$*
(i)	*all the variables bound in ϕ are mutually distinct*	

then we have: $(\mathcal{D},\mathcal{R}), \sigma \models \phi \;\Leftrightarrow\; [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^M (\!|\phi|\!)^{\mathbf{P}} \models [\![\phi]\!]^{\mathbf{Y}}$

Note that $[\![\sigma]\!]^M\downarrow \# (\!|\phi|\!)^{\mathbf{P}}\downarrow$*, since* $(\!|\phi|\!)^{\mathbf{P}}\downarrow = bv(\phi)$ *and* $bv(\phi) \# \sigma\downarrow$*,*
hence $[\![\sigma]\!]^M (\!|\phi|\!)^{\mathbf{P}} = (\!|\phi|\!)^{\mathbf{P}} [\![\sigma]\!]^M$*.*

Proof. By induction on ϕ and by cases.
Case $\exists x.\, \psi$

$[\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^M (\!|\exists x.\, \psi|\!)^{\mathbf{P}} \models [\![\exists x.\, \psi]\!]^{\mathbf{Y}}$
choose any $m' \in \Lambda \setminus \mathbf{P}$; let $\mathbf{P}' = \mathbf{P} \cup m'$
$\Leftrightarrow [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^M (\!|\psi|\!)^{\mathbf{P}'} \oplus \{x \leftarrow m'\} \models x \textcircled{\scriptsize R} ([\![\psi]\!]^{\mathbf{Y}\cup\{x\}} \wedge .x)$
$\qquad\qquad \vee \bigvee_{y\in\mathbf{Y}} [\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}}$
$\Leftrightarrow [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^M (\!|\psi|\!)^{\mathbf{P}'} \oplus \{x \leftarrow m'\} \models x \textcircled{\scriptsize R} ([\![\psi]\!]^{\mathbf{Y}\cup\{x\}} \wedge .x)$
$\quad \vee [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^M (\!|\psi|\!)^{\mathbf{P}'} \oplus \{x \leftarrow m'\} \models \bigvee_{y\in\mathbf{Y}} [\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}}$
By (h) and (i), x is not in the domain of σ or of $(\!|\psi|\!)^{\mathbf{P}'}$,
hence in the first line we can move $\oplus\{x \leftarrow m'\}$ to the term;
second line: remove $\oplus\{x \leftarrow m'\}$ since x does not appear in the formula
$\Leftrightarrow [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^M (\!|\psi|\!)^{\mathbf{P}'} \models (x \textcircled{\scriptsize R} ([\![\psi]\!]^{\mathbf{Y}\cup\{x\}} \wedge .x))\{x \leftarrow m'\}$
$\quad \vee [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^M (\!|\psi|\!)^{\mathbf{P}'} \models \bigvee_{y\in\mathbf{Y}} [\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}}$
We apply $\{x \leftarrow m'\}$ and expand $[\![\mathcal{D},\mathcal{R}]\!]^{M,N}$
$\Leftrightarrow (\vec{\nu}_{c\in N\downarrow} N(c))\, ([\![\mathcal{D}]\!]^{M\oplus N} \mid [\![\mathcal{R}]\!]^{M\oplus N}),$
$\qquad [\![\sigma]\!]^M (\!|\psi|\!)^{\mathbf{P}'} \models m' \textcircled{\scriptsize R} ([\![\psi]\!]^{\mathbf{Y}\cup\{x\}}\{x \leftarrow m'\} \wedge .m')$
$\quad \vee \exists y \in \mathbf{Y}.\, [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^M (\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}}$
Since $m' \notin fn([\![\mathcal{D},\mathcal{R}]\!]^{M,N})$ (by $\mathbf{P} \supseteq M\uparrow \supseteq fn([\![\mathcal{D},\mathcal{R}]\!]^{M,N})$), we apply
Corollary 5.2.6 (first line)
$\Leftrightarrow \exists c' \in N\downarrow .\, (\vec{\nu}_{c\in N\downarrow\setminus\{c'\}} N(c))\, (([\![\mathcal{D}]\!]^{M\oplus N} \mid [\![\mathcal{R}]\!]^{M\oplus N})\{N(c') \leftarrow m'\}),$
$\qquad [\![\sigma]\!]^M (\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi]\!]^{\mathbf{Y}\cup\{x\}}\{x \leftarrow m'\}$

$$\vee\ \exists y \in \mathbf{Y}.\ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M}(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}}$$

By Lemma 7.2.8 (1,2), since $M \oplus N \oplus \{c' \leftarrow m'\}$ is injective by $m' \notin \mathbf{P} \supseteq M \oplus N\!\uparrow$ and $N(c') = M \oplus N(c')$

$$\Leftrightarrow \exists c' \in N\!\downarrow\ .\ (\vec{\nu}_{c \in N\downarrow \setminus \{c'\}} N(c))\ [\![\mathcal{D}]\!]^{M \oplus N \oplus \{c' \leftarrow m'\}} \mid [\![\mathcal{R}]\!]^{M \oplus N \oplus \{c' \leftarrow m'\}},\ [\![\sigma]\!]^{M}(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi]\!]^{\mathbf{Y} \cup \{x\}}\{x \leftarrow m'\}$$

$$\vee\ \exists y \in \mathbf{Y}.\ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M}(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}}$$

By Lemma 7.2.8 (5), since $x \# bv(\psi)$ (i), $y \# bv(\psi)$ (h), and $x \notin \mathbf{Y}$ (h)

$$\Leftrightarrow \exists c' \in N\!\downarrow\ .\ (\vec{\nu}_{c \in N\downarrow \setminus \{c'\}} N(c))\ [\![\mathcal{D}]\!]^{M \oplus N \oplus \{c' \leftarrow m'\}} \mid [\![\mathcal{R}]\!]^{M \oplus N \oplus \{c' \leftarrow m'\}},\ [\![\sigma]\!]^{M}(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi]\!]^{\mathbf{Y} \cup \{x\}}\{x \leftarrow m'\}$$

$$\vee\ \exists y \in \mathbf{Y}.\ \ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, ([x \mapsto y]; [\![\sigma]\!]^{M})(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi]\!]^{\mathbf{Y}}$$

$y \in \sigma\!\downarrow$ and $x \notin \sigma\!\downarrow$, hence, by Lemma 7.2.8 (7,6)

$$\Leftrightarrow \exists c' \in N\!\downarrow\ .\ (\vec{\nu}_{c \in N\downarrow \setminus \{c'\}} N(c))\ [\![\mathcal{D}]\!]^{M \oplus N \oplus \{c' \leftarrow m'\}} \mid [\![\mathcal{R}]\!]^{M \oplus N \oplus \{c' \leftarrow m'\}},\ [\![\sigma]\!]^{M} \oplus \{x \leftarrow m'\}(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi]\!]^{\mathbf{Y} \cup \{x\}}$$

$$\vee\ \exists y \in \mathbf{Y}.\ \ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M} \oplus \{x \leftarrow M(\sigma(y))\}(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi]\!]^{\mathbf{Y}}$$

By Lemma 7.2.8 (4), since $c' \notin \sigma\!\uparrow$ by $N\!\downarrow \# M\!\downarrow$ (e) and $M\!\downarrow \supseteq \sigma\!\uparrow$ (b) and by Lemma 7.2.8 (3) (second line)

$$\Leftrightarrow \exists c' \in N\!\downarrow\ .\ (\vec{\nu}_{c \in N\downarrow \setminus \{c'\}} N(c))\ [\![\mathcal{D}]\!]^{M \oplus N \oplus \{c' \leftarrow m'\}} \mid [\![\mathcal{R}]\!]^{M \oplus N \oplus \{c' \leftarrow m'\}},\ [\![\sigma \oplus \{x \leftarrow c'\}]\!]^{M \oplus \{c' \leftarrow m'\}}(\!|\psi|\!)^{\mathbf{P}'} \not\# [\![\psi]\!]^{\mathbf{Y} \cup \{x\}}$$

$$\vee\ \exists y \in \mathbf{Y}.\ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma \oplus \{x \leftarrow \sigma(y)\}]\!]^{M}(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi]\!]^{\mathbf{Y}}$$

We now prepare the first line for the inductive step. We define $M' = M \oplus \{c' \leftarrow m'\}$, $N' = N \setminus c'$, $\mathbf{Y}' = \mathbf{Y} \cup \{x\}$, $\sigma' = \sigma \oplus \{x \leftarrow c'\}$. We check the induction conditions.

$$\begin{array}{ll}
(a) & \sigma'\!\downarrow\ =\ \sigma\!\downarrow \cup \{x\}\ \supseteq\ fv(\exists x.\,\psi) \cup \mathbf{Y} \cup \{x\}\ \supseteq\ fv(\psi) \cup \mathbf{Y}' \\
(b) & \sigma'\!\uparrow\ =\ \sigma\!\uparrow \cup \{c'\}\ \subseteq\ M\!\downarrow \cup \{c'\}\ =\ M'\!\downarrow \\
(c) & \sigma'(\mathbf{Y}')\ =\ \sigma(\mathbf{Y}) \cup \{c'\}\ =\ M\!\downarrow \cup \{c'\}\ =\ M'\!\downarrow \\
(d) & M'\!\downarrow \cup N'\!\downarrow\ =\ M\!\downarrow \cup \{c'\} \cup (N\!\downarrow \setminus \{c'\})\ =\ M\!\downarrow \cup N\!\downarrow\ =\ \mathcal{D} \\
(e) & M'\!\downarrow \# N'\!\downarrow \Leftrightarrow\ (M\!\downarrow \cup \{c'\}) \# (N\!\downarrow \setminus \{c'\})\ \Leftarrow\ M\!\downarrow \# N\!\downarrow \\
(f) & M'\!\uparrow \# N'\!\uparrow \Leftarrow\ (M\!\uparrow \cup \{m'\}) \# N\!\uparrow \Leftrightarrow\ (M\!\uparrow \# N\!\uparrow\ \wedge\ m' \notin N\!\uparrow) \\
(g) & \mathbf{P}'\ =\ \mathbf{P} \cup \{m'\}\ \supseteq\ (M \oplus N)\!\uparrow \cup \{m'\}\ \supseteq\ (M' \oplus N')\!\uparrow \\
(h) & bv(\psi) \# \sigma'\!\downarrow \Leftrightarrow\ (bv(\exists x.\,\psi) \setminus \{x\}) \# (\sigma\!\downarrow \cup \{x\})\ \Leftarrow\ bv(\exists x.\,\psi) \# \sigma\!\downarrow
\end{array}$$

Second line: $\Rightarrow$: let $c = \sigma(y)$. $\Leftarrow$ follows by $\sigma(\mathbf{Y}) = M\!\downarrow$

$$\Leftrightarrow \exists c' \in N\!\downarrow\ .\ (\vec{\nu}_{c \in N'\downarrow} N'(c))\ [\![\mathcal{D}]\!]^{M' \oplus N'} \mid [\![\mathcal{R}]\!]^{M' \oplus N'}, [\![\sigma']\!]^{M'}(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi]\!]^{\mathbf{Y}'}$$

$$\vee\ \exists c \in M\!\downarrow\ .\ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma \oplus \{x \leftarrow c\}]\!]^{M}(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi]\!]^{\mathbf{Y}}$$

By def. of $[\![\mathcal{D},\mathcal{R}]\!]^{M',N'}$

$$\Leftrightarrow \exists c' \in N\!\downarrow\ .\ [\![\mathcal{D},\mathcal{R}]\!]^{M',N'} [\![\sigma']\!]^{M'}(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi]\!]^{\mathbf{Y}'}$$

$$\vee\ \exists c \in M\!\downarrow\ .\ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma \oplus \{x \leftarrow c\}]\!]^{M}(\!|\psi|\!)^{\mathbf{P}'} \models [\![\psi]\!]^{\mathbf{Y}}$$

We now apply induction to both disjuncts. For the second line, we observe that (g) $\mathbf{P}' \supseteq M \oplus N$, (c) $(\sigma \oplus \{x \leftarrow c\})(\mathbf{Y}) = \sigma(\mathbf{Y}) = \sigma\uparrow$, (b) $(\sigma \oplus \{x \leftarrow c\})\uparrow = \sigma\uparrow \cup \{c\} \subseteq M\downarrow$.

$\Leftrightarrow \exists c' \in N\downarrow . (\mathcal{D}, \mathcal{R}), \sigma \oplus \{x \leftarrow c'\} \models \psi \ \vee \ \exists c \in M\downarrow . (\mathcal{D}, \mathcal{R}), \sigma \oplus \{x \leftarrow c\} \models \psi$

$\Leftrightarrow \exists c \in \mathcal{D}. (\mathcal{D}, \mathcal{R}), \sigma \oplus \{x \leftarrow c\} \models \psi$

$\Leftrightarrow (\mathcal{D}, \mathcal{R}), \sigma \models \exists x. \psi$

Cases $\neg\psi$, $\psi \vee \psi'$
Simple induction.

Case $R(x, x')$ Observe that, for any $c', c'' \in \mathcal{D}$:

$$(c', c'') \in \mathcal{R} \Leftrightarrow \exists T. \, [\![\mathcal{D}, \mathcal{R}]\!]^{M,N} = (\vec{\nu}_{c \in N\downarrow} N(c)) \, (M \oplus N(c')[M \oplus N(c'')[\mathbf{0}]] \mid T)$$

By $M\uparrow \# N\uparrow$, if $c', c'' \in M\downarrow$, the condition above is equivalent to:

$$(c', c'') \in \mathcal{R} \Leftrightarrow \exists T. \, [\![\mathcal{D}, \mathcal{R}]\!]^{M,N} \equiv M(c')[M(c'')[\mathbf{0}]] \mid (\vec{\nu}_{c \in N\downarrow} N(c)) \, T$$

i.e., for $c', c'' \in M\downarrow$:

$$\begin{aligned} & (c', c'') \in \mathcal{R} \\ \Leftrightarrow \ & \exists U. \, [\![\mathcal{D}, \mathcal{R}]\!]^{M,N} \equiv M(c')[M(c'')[\mathbf{0}]] \mid U \\ \Leftrightarrow \ & [\![\mathcal{D}, \mathcal{R}]\!]^{M,N} \models .M(c').M(c'') \end{aligned}$$

Hence:

$$\begin{aligned} & [\![\mathcal{D}, \mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M} (\!| R(x, x') |\!)^{\mathbf{P}} \models [\![R(x, x')]\!]^{\mathbf{Y}} \\ \Leftrightarrow \ & [\![\mathcal{D}, \mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M} \models .x.x' \\ \Leftrightarrow \ & [\![\mathcal{D}, \mathcal{R}]\!]^{M,N} \models .M(\sigma(x)).M(\sigma(x')) \\ \text{by } \sigma\uparrow \subseteq M\downarrow \ \Leftrightarrow \ & (\sigma(x), \sigma(x')) \in \mathcal{R} \\ \Leftrightarrow \ & \mathcal{D}, \mathcal{R}, \sigma \models R(x, x') \end{aligned}$$

□

Theorem 7.2.10. *For any closed FOL formula ϕ where all the free and bound variables are disjoint, for any $N : \mathcal{D} \xrightarrow{in} \Lambda$:*

$$\mathcal{D}, \mathcal{R} \models \phi \ \Leftrightarrow \ [\![\mathcal{D}, \mathcal{R}]\!]^{\emptyset, N} \models [\![\phi]\!]^{\emptyset} (\!| \phi |\!)^{\emptyset}$$

Proof. By Theorem 7.2.9, letting M be the empty function, $\mathbf{Y}$ be the empty set, $\mathbf{P} = N\uparrow$, and σ be the empty assignment, we have that

$$\mathcal{D}, \mathcal{R} \models \phi \ \Leftrightarrow \ [\![\mathcal{D}, \mathcal{R}]\!]^{\emptyset, N} \models [\![\phi]\!]^{\emptyset} [\![\emptyset]\!]^{\emptyset} (\!| \phi |\!)^{N\uparrow} \ \Leftrightarrow \ [\![\mathcal{D}, \mathcal{R}]\!]^{\emptyset, N} \models [\![\phi]\!]^{\emptyset} (\!| \phi |\!)^{N\uparrow}$$

$[\![\mathcal{D}, \mathcal{R}]\!]^{\emptyset, N} \models [\![\phi]\!]^{\emptyset} (\!| \phi |\!)^{N\uparrow}$ is equivalent to $[\![\mathcal{D}, \mathcal{R}]\!]^{\emptyset, N} \models [\![\phi]\!]^{\emptyset} (\!| \phi |\!)^{\emptyset}$ by Corollary 5.2.11, since $(\!| \phi |\!)^{\emptyset}$ and $(\!| \phi |\!)^{N\uparrow}$ are injective by construction, $(\!| \phi |\!)^{\emptyset}\downarrow = (\!| \phi |\!)^{N\uparrow}\downarrow = bv(\phi)$, and $fn([\![\mathcal{D}, \mathcal{R}]\!]^{\emptyset, N}, [\![\phi]\!]^{\emptyset})$ is empty. □

Corollary 7.2.11. *For any closed FOL formula ϕ where all the free and bound variables are disjoint $SAT_{FOL}(\phi) \Rightarrow SAT_{SL}([\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset})$*

Unfortunately, the inverse implication does not hold, because $[\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset}$ may be satisfied by SL models which are not the translation of any FOL model. Consider $(\exists x.\, \mathbf{T}) \wedge \neg(\exists y.\, \mathbf{T})$. It is clearly unsatisfiable, but it is translated (under $\mathbf{Y} = \emptyset$, $M = \emptyset$) as $m\textcircled{R}(\mathbf{T} \wedge .m) \wedge \neg n\textcircled{R}(\mathbf{T} \wedge .n)$, which is satisfied by the model $(\nu m')\, m'[\mathbf{0}] \mid n[\mathbf{0}]$, since the free occurrence of n prevents the model from satisfying $n\textcircled{R}(\mathbf{T} \wedge .n)$, while it satisfies $m\textcircled{R}(\mathbf{T} \wedge .m)$.

This fact does not contradict Theorem 7.2.10, since $(\nu m')\, m'[\mathbf{0}] \mid n[\mathbf{0}]$ is not the translation of any FOL model under $M = \emptyset$, because $[\![\mathcal{D}, \mathcal{R}]\!]^{\emptyset, N}$ has no free names. The fact that the model is not closed is actually the core of the problem since it may be possible that a formula in SAL is satisfied by not closed term only. We solve this problem by enriching the mapping with a conjunct that rules some of the non-closed models out.

Definition 7.2.12.

$$[\![\phi]\!]^{+} \stackrel{def}{=} [\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset} \wedge \bigwedge_{m \in nm([\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset})} \neg\, \textcircled{C}\, m$$

This new translation will ensure that any SL model of the translated formula is "closed enough", i.e. all its free names are disjoint from the names in the formula. Now we use the cut operation and Corollary 7.1.16 to show that these "residual" free names are irrelevant, hence that every model of the enriched translation actually corresponds to a FOL model, finally reducing SAT_{SL} to SAT_{FOL}.

Lemma 7.2.13. *Let $T = Cut_{N'}(U)$ for some N', U; then:*

$$fn(T) = \emptyset \;\Rightarrow\; \exists \mathcal{D}, \mathcal{R}, N.\; T = [\![\mathcal{D}, \mathcal{R}]\!]^{\emptyset, N}$$

Proof. $T = Cut_{T'}(\mathbf{P})$ implies that T is a set of two-lines $Par\{m_i[m_i'[\mathbf{0}]] : i \in I\}$ plus a set of one-lines $Par\{n_j[\mathbf{0}] : j \in J\}$, with the property that $\forall i \in I.\ m_i \in \{n_j\}^{j \in J}, m_i' \in \{n_j\}^{j \in J}$. This set of one- and two-lines is preceded by a string of restrictions. $fn(T) = \emptyset$ implies that every name in T is actually restricted, i.e. that:

$$T = (\vec{\nu}_{j \in J} n_j)\, Par\{m_i[m_i'[\mathbf{0}]] : i \in I\} \mid Par\{n_j[\mathbf{0}] : j \in J\}$$

Now, the thesis follows by choosing $\mathcal{D} = \{n_j\}^{j \in J}$, $\mathcal{R} = \{(m_i, m_i')\}^{i \in I}$, and letting N be the identity function over $\mathcal{D}$. □

Theorem 7.2.14 (Reduction of FOL Satisfiability)**.** *For any closed FOL formula ϕ, $SAT_{FOL}(\phi) \Leftrightarrow SAT_{SL}([\![\phi]\!]^{+})$*

Proof. ($\Rightarrow$) Let $\mathcal{D}, \mathcal{R}$ be such that $(\mathcal{D}, \mathcal{R}), \emptyset \models \phi$. By Corollary 7.2.10, $[\![\mathcal{D}, \mathcal{R}]\!]^{\emptyset, N}$ satisfies $[\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset}$. Since $[\![\mathcal{D}, \mathcal{R}]\!]^{\emptyset, N}$ is closed, it also satisfies $\neg\, \textcircled{C}\, m$ for any m.

($\Leftarrow$) Assume $SAT_{SL}([\![\phi]\!]^+)$. Then, there exists T such that:

$$T \models [\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset} \quad (1)$$
$$T \models \bigwedge_{m \in nm([\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset})} \neg \textcircled{c}\, m \quad (2)$$

Consider now $U = Cut_{nm([\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset})}(T)$. By Lemma 7.1.14:

$$U \models [\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset} \quad (3)$$
$$U \models \bigwedge_{m \in nm([\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset})} \neg \textcircled{c}\, m \quad (4)$$

By Lemma 7.1.7

$$fn(U) \subseteq nm([\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset})$$

by (4)

$$fn(U) \# nm([\![\phi]\!]^{\emptyset}(\!|\phi|\!)^{\emptyset})$$

hence

$$fn(U) = \emptyset$$

By Lemma 7.2.13, U is the translation of a FOL interpretation $\mathcal{D}, \mathcal{R}$. □

7.3 Encoding Hiding in FOL

The proof of undecidability of hiding quantification is very similar to that of revelation.

The translation is slightly simpler since we do not need the $(\!|\phi|\!)^{\mathbf{P}}$ substitution any more. Moreover, since the translation of a formula contains no free name, the step from the faithfulness theorem to the undecidability corollary is shorter as well.

Definition 7.3.1 (Formula translation)**.** *FOL formulas, assignments, interpretations, domains, and relations, are translated as in Definition 7.2.7, under the same conditions over ϕ, $\mathbf{Y}$, Λ, M, and N, with the only exception of the existential quantifier, as specified below. This time we need no translation of formulas into substitutions, hence we do not need the set $\mathbf{P}$.*

formulas

$$[\![\exists x.\, \phi]\!]^{\mathbf{Y}} \stackrel{def}{=} \mathbf{H}x.\, ([\![\phi]\!]^{\mathbf{Y} \cup \{x\}} \wedge .x) \vee \bigvee_{y \in \mathbf{Y}} [\![\phi\{x \leftarrow y\}]\!]^{\mathbf{Y}}$$

...

Lemma 7.3.2. *If $\{x, y\} \# bv(\phi)$ and $x \notin \mathbf{Y}$, then:* $[\![\phi\{x \leftarrow y\}]\!]^{\mathbf{Y}} = [\![\phi]\!]^{\mathbf{Y}}\{x \leftarrow y\}$

Proof. By induction and by cases. Case $\phi = \exists z.\ \psi$:

$$
\begin{array}{ll}
[\![(\exists z.\ \psi)\{x \leftarrow y\}]\!]^{\mathbf{Y}} & \text{by } x, y \neq z \\
= [\![\exists z.\ \psi\{x \leftarrow y\}]\!]^{\mathbf{Y}} & \text{by def.} \\
= \mathbf{H}z.\, ([\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}\cup\{z\}} \wedge .z) \vee \bigvee_{w\in\mathbf{Y}} [\![\psi\{x \leftarrow y\}\{z \leftarrow w\}]\!]^{\mathbf{Y}} & \text{by } x, y \neq z \\
 & w \neq x \\
= \mathbf{H}z.\, ([\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}\cup\{z\}} \wedge .z) \vee \bigvee_{w\in\mathbf{Y}} [\![\psi\{z \leftarrow w\}\{x \leftarrow y\}]\!]^{\mathbf{Y}} & \text{ind.} \\
 & (x \notin \mathbf{Y} \cup \{z\}) \\
= \mathbf{H}z.\, ([\![\psi]\!]^{\mathbf{Y}\cup\{z\}}\{x \leftarrow y\} \wedge .z) \vee \bigvee_{w\in\mathbf{Y}} [\![\psi\{z \leftarrow w\}]\!]^{\mathbf{Y}}\{x \leftarrow y\} & \text{by } x \neq z \\
= (\mathbf{H}z.\, ([\![\psi]\!]^{\mathbf{Y}\cup\{z\}} \wedge .z) \vee \bigvee_{w\in\mathbf{Y}} [\![\psi\{z \leftarrow w\}]\!]^{\mathbf{Y}})\{x \leftarrow y\} & \text{by def.} \\
= [\![\exists z.\ \psi]\!]^{\mathbf{Y}}\{x \leftarrow y\}
\end{array}
$$

□

Theorem 7.3.3 (Faithfulness of translation)**.** *For any FOL formula ϕ, for any interpretation $(\mathcal{D}, \mathcal{R})$, for any set of variables $\mathbf{Y}$, for any variable assignment σ, for any pair of partial injective functions M,$N : \mathcal{D} \xrightarrow{in} \Lambda$, such that:*

(a)	$\sigma\downarrow \supseteq fv(\phi) \cup \mathbf{Y}$	*σ closes the free variables of ϕ and those in $\mathbf{Y}$*
(b)	$\sigma\uparrow \subseteq M\downarrow$	*σ sends everything into the M-elements*
(c)	$\sigma(\mathbf{Y}) = M\downarrow$	*every M-element is reached by σ*
(d)	$M\downarrow \cup N\downarrow = \mathcal{D}$	*we know how to translate any $c \in \mathcal{D}$*
(e)	$M\downarrow \,\#\, N\downarrow$	*M-elements and N-elements are distinct*
(f)	$M\uparrow \,\#\, N\uparrow$	*M-names and N-names are distinct*
(g)	$bv(\phi) \# \sigma\downarrow$	*$bv(\phi) \# fv(\phi)$ and $bv(\phi) \# \mathbf{Y}$*
(h)	*all the variables bound in ϕ are mutually distinct*	

then we have: $(\mathcal{D}, \mathcal{R}), \sigma \models \phi \;\Leftrightarrow\; [\![\mathcal{D}, \mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M} \models [\![\phi]\!]^{\mathbf{Y}}$

Proof. By induction on ϕ and by cases, along the lines of Theorem 7.2.9.
Case $\exists x.\ \psi$

$$
\begin{array}{l}
[\![\mathcal{D}, \mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M} \models [\![\exists x.\ \psi]\!]^{\mathbf{Y}} \\
\Leftrightarrow [\![\mathcal{D}, \mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M} \models \mathbf{H}x.\, ([\![\psi]\!]^{\mathbf{Y}\cup\{x\}} \wedge .x) \\
\qquad\qquad\qquad\qquad \vee\ \bigvee_{y\in\mathbf{Y}} [\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}} \\
\Leftrightarrow [\![\mathcal{D}, \mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M} \models \mathbf{H}x.\, ([\![\psi]\!]^{\mathbf{Y}\cup\{x\}} \wedge .x) \\
\quad \vee\ [\![\mathcal{D}, \mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M} \models \bigvee_{y\in\mathbf{Y}} [\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}} \\
\Leftrightarrow (\vec{\nu}_{c\in N\downarrow} N(c))\, [\![\mathcal{D}]\!]^{M\oplus N} \mid [\![\mathcal{R}]\!]^{M\oplus N}, [\![\sigma]\!]^{M} \models \mathbf{H}x.\, ([\![\psi]\!]^{\mathbf{Y}\cup\{x\}} \wedge .x) \\
\quad \vee\ [\![\mathcal{D}, \mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M} \models \bigvee_{y\in\mathbf{Y}} [\![\psi\{x \leftarrow y\}]\!]^{\mathbf{Y}}
\end{array}
$$

Choose $m' \notin M \oplus N\uparrow$; this makes it fresh enough to apply Corollary 5.2.8

$$
\begin{array}{l}
\Leftrightarrow \exists c' \in N\downarrow .\ (\vec{\nu}_{c\in N\downarrow\setminus\{c'\}} N(c))\, (([\![\mathcal{D}]\!]^{M\oplus N} \mid [\![\mathcal{R}]\!]^{M\oplus N})\{N(c') \leftarrow m'\}), \\
\qquad\qquad\qquad\qquad [\![\sigma]\!]^{M} \qquad\qquad\qquad\qquad \models [\![\psi]\!]^{\mathbf{Y}\cup\{x\}}\{x \leftarrow m'\}
\end{array}
$$

$$\vee\ \exists y{\in}\mathbf{Y}.\ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M} \models [\![\psi\{x{\leftarrow}y\}]\!]^{\mathbf{Y}}$$

By Lemma 7.2.8 (1,2), since $M \oplus N(c') = N(c')$ and $M \oplus N : M \oplus N \xrightarrow{in} \Lambda$

$$\Leftrightarrow \exists c'{\in}N{\downarrow}\ .\ (\vec{\nu}_{c\in N{\downarrow}\setminus\{c'\}}N(c))\ [\![\mathcal{D}]\!]^{M\oplus N\oplus\{c'\leftarrow m'\}} \mid [\![\mathcal{R}]\!]^{M\oplus N\oplus\{c'\leftarrow m'\}},\ [\![\sigma]\!]^{M} \models [\![\psi]\!]^{\mathbf{Y}\cup\{x\}}\{x{\leftarrow}m'\}$$
$$\vee\ \exists y{\in}\mathbf{Y}.\ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M} \models [\![\psi\{x{\leftarrow}y\}]\!]^{\mathbf{Y}}$$

By Lemma 7.3.2 ($x \notin bv(\psi)$ by (h); $y \notin bv(\psi)$ by $\mathbf{Y}\#\, bv(\exists x.\,\psi)$; $x \notin \mathbf{Y}$ by $\mathbf{Y}\#\, bv(\exists x.\,\psi)$)

$$\Leftrightarrow \exists c'{\in}N{\downarrow}\ .\ (\vec{\nu}_{c\in N{\downarrow}\setminus\{c'\}}N(c))\ [\![\mathcal{D}]\!]^{M\oplus N\oplus\{c'\leftarrow m'\}} \mid [\![\mathcal{R}]\!]^{M\oplus N\oplus\{c'\leftarrow m'\}},\ [\![\sigma]\!]^{M} \models [\![\psi]\!]^{\mathbf{Y}\cup\{x\}}\{x{\leftarrow}m'\}$$
$$\vee\ \exists y{\in}\mathbf{Y}.\ \ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [x{\mapsto}y]; [\![\sigma]\!]^{M} \models [\![\psi]\!]^{\mathbf{Y}}$$

$y{\in}\sigma{\downarrow}$ and $x{\notin}\sigma{\downarrow}$, hence, by Lemma 7.2.8 (7,6)

$$\Leftrightarrow \exists c'{\in}N{\downarrow}\ .\ (\vec{\nu}_{c\in N{\downarrow}\setminus\{c'\}}N(c))\ [\![\mathcal{D}]\!]^{M\oplus N\oplus\{c'\leftarrow m'\}} \mid [\![\mathcal{R}]\!]^{M\oplus N\oplus\{c'\leftarrow m'\}},\ [\![\sigma]\!]^{M}\oplus\{x{\leftarrow}m'\} \models [\![\psi]\!]^{\mathbf{Y}\cup\{x\}}$$
$$\vee\ \exists y{\in}\mathbf{Y}.\ \ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma]\!]^{M}\oplus\{x{\leftarrow}M(\sigma(y))\} \models [\![\psi]\!]^{\mathbf{Y}}$$

By Lemma 7.2.8 (4), since $c' \notin \sigma{\uparrow}$ by $N{\downarrow}\,\#\,M{\downarrow}$ (e) and $M{\downarrow}{\supseteq}\,\sigma{\uparrow}$ (b) and by Lemma 7.2.8 (3) (second line)

$$\Leftrightarrow \exists c'{\in}N{\downarrow}\ .\ (\vec{\nu}_{c\in N{\downarrow}\setminus\{c'\}}N(c))\ [\![\mathcal{D}]\!]^{M\oplus N\oplus\{c'\leftarrow m'\}} \mid [\![\mathcal{R}]\!]^{M\oplus N\oplus\{c'\leftarrow m'\}},\ [\![\sigma\oplus\{x{\leftarrow}c'\}]\!]^{M\oplus\{c'\leftarrow m'\}} \models [\![\psi]\!]^{\mathbf{Y}\cup\{x\}}$$
$$\vee\ \exists y{\in}\mathbf{Y}.\ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma\oplus\{x{\leftarrow}\sigma(y)\}]\!]^{M} \models [\![\psi]\!]^{\mathbf{Y}}$$

We now prepare the first line for the inductive step. We define $M' = M \oplus \{c'{\leftarrow}m'\}$, $N' = N \setminus c'$, $\mathbf{Y}' = \mathbf{Y}\cup\{x\}$, $\sigma' = \sigma\oplus\{x{\leftarrow}c'\}$. We check the induction conditions.

$$\begin{array}{ll}
(a) & \sigma'{\downarrow} = \sigma{\downarrow}\cup\{x\} \supseteq fv(\exists x.\,\psi)\cup\mathbf{Y}\cup\{x\} \supseteq fv(\psi)\cup\mathbf{Y}'\\
(b) & \sigma'{\uparrow} = \sigma{\uparrow}\cup\{c'\} \subseteq M{\downarrow}\cup\{c'\} = M'{\downarrow}\\
(c) & \sigma'(\mathbf{Y}') = \sigma(\mathbf{Y})\cup\{c'\} = M{\downarrow}\cup\{c'\} = M'{\downarrow}\\
(d) & M'{\downarrow}\cup N'{\downarrow} = M{\downarrow}\cup\{c'\}\cup(N{\downarrow}\setminus\{c'\}) = M{\downarrow}\cup N{\downarrow} = \mathcal{D}\\
(e) & M'{\downarrow}\,\#\,N'{\downarrow} \Leftrightarrow (M{\downarrow}\cup\{c'\})\#\,(N{\downarrow}\setminus\{c'\}) \Leftarrow M{\downarrow}\,\#\,N{\downarrow}\\
(f) & M'{\uparrow}\,\#\,N'{\uparrow} \Leftarrow (M{\uparrow}\cup\{m'\})\#\,N{\uparrow} \Leftrightarrow (M{\uparrow}\,\#\,N{\uparrow}\ \wedge\ m'\notin N{\uparrow})\\
(g) & bv(\psi)\#\,\sigma'{\downarrow} \Leftrightarrow (bv(\exists x.\,\psi)\setminus\{x\})\#\,(\sigma{\downarrow}\cup\{x\}) \Leftarrow bv(\exists x.\,\psi)\#\,\sigma{\downarrow}
\end{array}$$

Second line: $\Rightarrow$: let $c = \sigma(y)$. $\Leftarrow$ follows by $\sigma(\mathbf{Y}) = M{\downarrow}$

$$\Leftrightarrow \exists c'{\in}N{\downarrow}\ .\ (\vec{\nu}_{c\in N'{\downarrow}}N'(c))\ [\![\mathcal{D}]\!]^{M'\oplus N'} \mid [\![\mathcal{R}]\!]^{M'\oplus N'}, [\![\sigma\oplus\{x{\leftarrow}c'\}]\!]^{M'} \models [\![\psi]\!]^{\mathbf{Y}'}$$
$$\vee\ \exists c{\in}M{\downarrow}\ .\ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma\oplus\{x{\leftarrow}c\}]\!]^{M} \models [\![\psi]\!]^{\mathbf{Y}}$$

By def. of $[\![\mathcal{D},\mathcal{R}]\!]^{M',N'}$

$$\Leftrightarrow \exists c'{\in}N{\downarrow}\ .\ [\![\mathcal{D},\mathcal{R}]\!]^{M',N'}[\![\sigma\oplus\{x{\leftarrow}c'\}]\!]^{M'} \models [\![\psi]\!]^{\mathbf{Y}'}$$
$$\vee\ \exists c'{\in}M{\downarrow}\ .\ [\![\mathcal{D},\mathcal{R}]\!]^{M,N}, [\![\sigma\oplus\{x{\leftarrow}c'\}]\!]^{M} \models [\![\psi]\!]^{\mathbf{Y}}$$

We now apply induction to both disjuncts.

$\Leftrightarrow \exists c' \in N\downarrow . (\mathcal{D},\mathcal{R}), \sigma \oplus \{x \leftarrow c'\} \models \psi \ \vee \ \exists c' \in M\downarrow . (\mathcal{D},\mathcal{R}), \sigma \oplus \{x \leftarrow c'\} \models \psi$
$\Leftrightarrow \exists c \in \mathcal{D}. (\mathcal{D},\mathcal{R}), \sigma \oplus \{x \leftarrow c\} \models \psi$
$\Leftrightarrow (\mathcal{D},\mathcal{R}), \sigma \models \exists x. \psi$

Cases $\neg\psi$, $\psi \vee \psi'$, $R(x,y)$: As in the proof of Theorem 7.2.9.

□

Lemma 7.3.4 (Equivalence of satisfiability)**.** *For any closed FOL formula* ϕ, $SAT_{FOL}(\phi) \Leftrightarrow SAT_{SL}([\![\phi]\!]^{\emptyset})$

Proof. ($\Rightarrow$) Assume all variables in ϕ are distinct. Let $\mathcal{D},\mathcal{R}$ be such that $(\mathcal{D},\mathcal{R}),\emptyset \models \phi$. Consider any $N : \mathcal{D} \xrightarrow{in} \Lambda$. By Theorem 7.3.3, $[\![\mathcal{D},\mathcal{R}]\!]^{\emptyset,N}$ satisfies $[\![\phi]\!]^{\emptyset}$ (as in the proof of Corollary 7.2.10).

($\Leftarrow$) Assume $SAT_{SL}([\![\phi]\!]^{\emptyset})$. Then, there exists T such that:

$$T \models [\![\phi]\!]^{\emptyset}$$

Consider now $U = Cut_{nm([\![\phi]\!]^{\emptyset})}(T) = Cut_{\emptyset}(T)$. By Lemma 7.1.14, $U \models [\![\phi]\!]^{\emptyset}$. By Lemma 7.1.7, $fn(U) = \emptyset$. By Lemma 7.2.13, $U = [\![\mathcal{D},\mathcal{R}]\!]^{\emptyset,N}$ for some $\mathcal{D}$, $\mathcal{R}$, N. By Theorem 7.3.3, $U = [\![\mathcal{D},\mathcal{R}]\!]^{\emptyset,N} \models [\![\phi]\!]^{\emptyset}$ implies $\mathcal{D},\mathcal{R} \models \phi$; hence $SAT_{FOL}(\phi)$. □

7.4 Undecidability Results

Corollary 7.4.1 (Undecidability of revelation)**.** *Satisfiability (hence validity) of closed formulas built from* $n \textcircled{R} A$, $A \wedge A$, $\neg A$, $.n$, $.n_1.n_2$, *is not decidable.*

Proof. This is a corollary of Lemma 7.2.14. Just observe that, while open paths like $.x.y$ may appear in the translation of a generic formula ϕ, they never appear in the translation of a closed formula. □

Corollary 7.4.2 (Undecidability of Hiding)**.** *Satisfiability (hence validity) of closed formulas built from* $\mathbf{H}x.\, A$, $A \wedge A$, $\neg A$, $.x_1$, *and* $.x_1.x_2$, *is not decidable.*

Proof. Follows from the reduction into FOL satisfiability on finite domanis that is undecidable. □

7.5 Classification

In SL hiding can be expressed as freshness plus revelation. The main result proved in this Chapter can be summarized as: freshness without revelation gives a rich decidable logic (Corollary 6.3.3) while revelation makes a minimal logic undecidable (Corollary 7.4.1). We also proved that hiding is undecidable, and some results about extrusion that we summarize below.

Table 7.5.1. *A summary of decidability/extrusion results*

Logic	Decidable?	Op	Extrusion algorithm
$SL_{\{\}}$	Yes, proved in [32]	$\mathsf{И}$	Yes, see Table 7.5.6.2 and [90]
$SL_{\{\mathsf{И},\circledcirc,\circledcirc\}}$	Yes, proved in Corollary 6.3.3	Ⓡ	No, by Corollary 6.3.4
$SL_{\{\exists\}}$	No, follows from [50]	**H**	No, by Corollary 6.3.4
$SL_{\{@\}}$	No, follows from Corollary 7.4.1	$\exists$	No, by Corollary 6.3.4
$SL_{\{\mathsf{H}\}}$	No, follows from Corollary 7.4.2		

The decidability result is based on the extrusion of freshness into a prenex form. The proof of decidability by extrusion is very attractive because it does not need combinatorial explorations of the model, but is based on the "algebraic" properties of the logic, and is robust with respect to variations on the logic itself.

The undecidable logic is obtained by adding revelation to a minimal logic of propositional connectives and simple path formulas, hence we show that undecidability comes from revelation and not from the spatial nature of SL. Undecidability of any richer logic follows immediately.

7.6 Related Work

Independently of this study, an extrusion algorithm for the freshness quantifier (Section 6.2) has been developed in [90] by Lozes. The main result of that paper is a surprising adjunct elimination theorem for $SL_{\{\mathsf{И},@,\oslash\}}$.

The result is surprising in view of the fact that the parallel-adjunct seems to be extremely expressive, being able to quantify over infinite sets of trees, and of internalizing validity into model-checking.

Lozes leaves the open problem of the existence of an effective adjunct-elimination procedure. As a corollary of our undecidability results, we can close that problem.

Corollary 7.6.1. *No effective adjunct-elimination exists for the logic* $SL_{\{\mathsf{И},@,\oslash\}}$.

A calculus to manipulate trees with hidden names has been presented in [38], whose type system includes the full SL. As a result, type inclusion in that calculus and validity in SL are mutually reducible. Decidability of subtype-checking was left as an open problem in [38]. Our results imply that it is undecidable.

Part IV

An Application: Spatial Logics for Web Data

Chapter 8

Web Data Overview

In the era of Web services and Web applications there is tremendous need for database-like functionality to efficiently provide and access data on the Web. But a classical database is a coherently designed system that imposes rigid structure, and provides queries, updates, as well as transactions, concurrency, integrity, and recovery, in a controlled environment. The Web escapes any such control. Data on the Web is free-evolving and ever-changing, and it has various shapes and forms. For these reasons much of the traditional framework of database theory needs to be reinvented in the Web scenario. The database community is performing an intensive work in this direction. The self-describing and irregular structure of data on the Web has been formalized by *semi-structured data* (SSD for short). Several schema definition languages for SSD and XML have been proposed. The problem of *constraint specification* (generalization of the classical dependencies to the SSD and XML framework) has been addressed. Many query languages for semi-structured data and XML have been studied, but their expressivity is not easy to characterize. The complexity of queries is also hard to evaluate.

Understanding interaction of schema, constraints and queries for SSD and XML is a very important issue that encloses several current topics of research. For this reason a formal environment that combines constraints, types and query expressions is an interesting perspective. Using this unified formal framework we can reason about:

Schema vs. constraints Types and constraints decision problems (and their interactions) can be studied in a unified view of schema and constraints. In particular it is possible to study the impact of schema formalism on standard constraint decision problems.

Constraints vs. queries Query optimization guided by constraints can be formalized.

Schema vs. queries A static type-system can be obtained by combining query and type expressions.

The main problem of such an environment is the expressive power of the formalism we want to use. There is the usual trade-off between expressivity of the formalism and decidability of the resulting language.

A well-studied candidate formalism is the ambient logic [43, 45]. The idea of using a spatial tree logic to describe properties of SSD is due to previous works of Cardelli and Ghelli [42, 37] and to the know-how we gained during the implementation of the TQL query language [58, 54].

TQL is a query language for SSD that uses spatial tree logic formulas to express properties of data that will be collected using a pattern-matching mechanism. The logic used in TQL is very expressive and allows us to express complex types, constraints, and queries, giving us, for types and constraints, an expressive power that is higher than the one of other proposals [76, 20]. Of course, if the full power of the logic is used, every aspect of static query analysis (correctness, containment, subtyping...) becomes undecidable, since validity of a tree-logic formula is undecidable in general. On the other side, many decidable subsets of the logic can be defined, which are expressive enough to encode known type and constraint systems. The search for decidable subsets with the "right" balance of expressivity and cost is a delicate problem. As happens with first order logic, undecidability of validity/satisfaction does not hinder the usability of the logic as a tool to query a database, since this usage is related to the decidable problem of whether a formula holds in just one specific model (or database).

8.1 Motivating example

To show the advantages provided by a language capable to express queries, constraints and types over semistructured data, we present a motivating example. The scenario includes a semi-structured data source D (i.e. an XML file) on the Web, a schema S (i.e. a DTD or an XML Schema), and a set of integrity constraints C (i.e. some inclusion and key constraint) specified in some constraint language (i.e. path inclusion language or key constraint language).

Suppose to translate S in an equivalent[1] spatial tree logic formula A_S and similarly C in another formula A_C. The unsatisfiability of A_S corresponds to the well-known problem of *schema emptiness*, and the satisfiability of A_C corresponds to the problem of *constraint consistency*. The satisfiability of the legitimate logic formula $A_S \wedge A_C$ corresponds to another interesting and little investigated problem: *constraint consistency in presence of a schema*. Other useful schema decision problems that could be investigated in an unified logic include *schema equivalence*, *schema inclusion* and *schema disjointness*.

Another classical decision problem is the *constraint implication*: given that some constraints are known to hold, does it follow that some other constraint is necessar-

[1]Since our current logic is unordered, all the reasonings (i.e. equivalence) we refer to are up to document order.

ily satisfied? This corresponds in our logic to the validity of an implication formula $A_C \Rightarrow A'_C$. Similarly the *constraint implication in presence of schema* corresponds to a $A_S \wedge A_C \Rightarrow A'_C$ where A_S specifies the schema, and A_C and A'_C are constraints. Constraint implication is important, among other things, in data integration, database normalization and, as we will see, in query optimization.

The *validation problem* of the data source D w.r.t a schema S and a set of constraints C becomes, in our logic environment, a satisfaction problem of the form $D \models A_S \wedge A_C$ in the ambient logic style. A generalization of this satisfaction relation dealing with free variables is the core of TQL binding mechanism. This allow us to perform checking of data properties (i.e. validation of constraints) by executing TQL yes/no queries encoding the satisfaction relation. Of course the actual cost of yes/no query execution is not comparable with the cost of algorithms studied for specific validation problems (see for example [36]), but the combination of TQL query execution model with well-studied standard structures for optimal validation is an interesting possible research.

Our scenario includes also the consumer side of the Web: a user that queries the data source D. The user may ignore schema and constraints over the data source. Suppose that the user expresses his query with a tree logic formula B, a static analysis on $A_S \wedge B$ can be performed to check whether the query conforms to the schema. If that formula is unsatisfiable, we can statically state the emptiness of the query result avoiding query execution. In a similar way we can use constraints on D that are known to hold, eventually combined with the schema.

Reasoning on constraints, type and queries can be used as a query optimization tool. As a trivial example we have the following query binding expression: "Bind all the books in the bibliography file that contain at least an author element to X". If we know by the schema (or by a known constraint) that "All books have at least an author", the previous query is equivalent to the cheaper one "Bind all the books in the bibliography file to X".

More interesting examples are optimizations based on key constraint implications and combinations of key constraints and inclusion constraints with the schema. More generally, given the formulas A_S, A_C, and B representing respectively the logic formalization of the schema, the set of validated constraints, and the query, we can substitute B with every cheaper query B' such that $A_S \wedge A_C \Rightarrow (B \Leftrightarrow B')$. Of course the query execution cost depends upon the physical layer and the available index structures. Therefore a complete optimization tool should combine such logical rewritings with standard physical optimization techniques.

8.2 Semistructured data and XML

Semistructured data [19] is a bare-bones abstraction of the irregular, self-describing data found on the Web. It is also motivated by applications such as scientific databases, and the integration of heterogeneous data. The semi-structured data model

is made of labeled graphs. The nodes are viewed as objects and have object ids. Objects can be atomic or complex: complex objects are linked to other objects by labeled edges, and atomic objects contain data values. Several variants of the semistructured data model have been proposed, with minor differences in formalism [51, 22, 6].

While SSD originated in the database community, the XML [17] (Extended Markup Language) has been introduced in the document community as a subset of SGML. An XML document consists of nested elements, with ordered sub-elements. The simplest abstraction of XML data is a labeled ordered tree (with labels on nodes), possibly with data values associated to the leaves. In Figure 8.1 an example of an XML document is presented.

XML files can be easily translated into equivalent (up to document order) information trees. As an example, the XML file in Figure 8.1 can be represented by the information tree of Figure 8.2

XML additionally provides a referencing mechanism among elements that allows simulating arbitrary graphs, and so SSD. But this aspect is left out of some formal models, because neither XML schema nor query languages takes advantage of it.

A compact survey on the SSD and XML research status is [111], the book [5] is an invaluable source of information on databases and the Web.

8.2.1 Semistructured Types and Constraints

Although semistructured data and XML are self-describing and thus do not require any schema, constraint or type system, such systems are known to be useful to more efficiently process, query, and manage data.

In SSD and XML context the distinction between data *types* (or schema) and *constraints* is blurred. This is because a tree (or graph) interpretation of the data is used, and both traditional types and constraints can be viewed as constraints over the structure of this interpretation. Nevertheless schema and constraints remain separated concepts from the user perspective. In [23] it was observed that the distinction between types and constraints is dictated largely by what conventional programming languages treat as types. In XML there is another possible distinction regarding constrained objects. A type (or schema) is a constraint on the structure of the documents (i.e. path of tag elements), while integrity constraints regard the *values* (leaves of the data tree).

Schema languages for XML

The XML formalism includes a notion of schema specified in form of Data Type Definitions (DTDs). Essentially a DTD is an extended context-free grammar, with labels as non-terminals and with no terminal symbols. More formally, given the set of labels Λ, a DTD consists of a set of rules of the form $L \rightarrow R$ where $L \in \Lambda$ and R is a regular expression over Λ. One rule must be defined for each L, and the

```
<bib>
  <book>
    <year>1999</year>
    <title>Foundations of Databases</title>
    <author>S. Abiteboul</author>
    <author>R. Hull</author>
    <author>V. Vianu</author>
    <publisher>Addison-Wesley</publisher>
  </book>
  <article>
    <year>2001</year>
    <title>A Web odyssey: from Codd to XML</title>
    <author>
      <name> Victor </name>
      <surname> Vianu </surname>
    </author>
    <booktitle>Proc. of PODS 2001</booktitle>
    <series>SIGMOD Record</series>
  </article>
  <article>
    <year></year>
    <title>Ambient Logic</title>
    <author>L. Cardelli</author>
    <author>A. Gordon</author>
    <note>Submitted for publication</note>
  </article>
</bib>
```

Figure 8.1: An example of an XML file describing a bibliography. This file can be interpreted as a tree labeled `bib` with sub-trees labeled respectively `book`, `article`, and `article`.

```
bib[
  book[
    year[1999] |
    title[Foundations of Databases] |
    author[S. Abiteboul] |
    author[R. Hull] |
    author[V. Vianu] |
    publisher[Addison-Wesley]
  ] |
  article[
    year[2001] |
    title[A Web odyssey: from Codd to XML] |
    author[
      name[Victor] |
      surname[Vianu]
    ] |
    booktitle[Proc. of PODS 2001]|
    series[SIGMOD Record]
  ] |
  article[
    year[0] |
    title[Ambient Logic] |
    author[L. Cardelli] |
    author[A. Gordon] |
    note[Submitted for publication]
  ]
]
```

Figure 8.2: An information tree describing a bibliography. It represents the same data of Figure 8.1 up to document order.

DTD also specifies the label of the root. An XML document that is a derivation of its DTD grammar is *valid*. For example, a DTD might consist of the rules (labels without rules correspond to strings):

$root : section;$

$section \rightarrow intro, section*, conc$

An XML file satisfying the above DTD is:

```
<section>
  <intro> Introduction <\intro>
  <section>
    <intro> Intro1 <\intro>
```

```
    <section>
       <intro> Intro1.1 <\intro>
       <conc><\conc>
    <\section>
    <conc><\conc>
  <\section>
  <conc> The End. <\conc>
<\section>
```

Thus, each DTD d defines a set of ordered trees $val(d)$. It turns out that DTDs have many limitations as schema languages (i.e. inability to separate the concepts of *type* and *name* of an element, lack of flexibility in specifying ordering constraints). Decoupling element names from their types is formalized using *specialized DTD*(studied in [102]). The idea is to use "specializations" of element names whenever necessary, with their own type definition.

More recently, many schema languages and language-based schema systems extending DTDs have been proposed, including XML Schema [2], DSD [82], RELAX NG [52]. An interesting approach to generalize DTDs is the *regular expression types* proposal of [77] used in the design of the XML processing language XDuce [76]. Given the set of labels Λ ranged over by l and a set of type variables ranged over by X, type expressions are defined as follows:

Table 8.2.1. *Regular expression types*

$T, U ::=$	type expression
$()$	empty sequence
X	variable
$l[T]$	label
T, U	concatenation
$T \mid U$	union

The regular expression operators *, +, and ? are derivable as combinations of the above constructors.

Regular expression types captures the regular expression notations commonly found in schema languages for XML, and support natural "semantic" notion of subtyping. They can be used for static typechecking in a language for processing XML.

Comparative analysis of several XML schema languages can be found in [5] and [87].

Integrity Constraints

Integrity constraints are essential ingredients of classical databases for many purposes, including invalid data filtering, schema design, and query optimizations. Not

surprisingly, they continue to be important in semistructured data and XML. However, in SSD the properties to be granted and the way constraints are specified is very different from databases. In [23] traditional integrity constraints for SSD are classified in *inclusion constraints* (i.e. "all students are people"), *inverse constraints* (i.e. the `taken` relationship from students to courses is the inverse of `taken_by` from courses to students), and *key constraints*(i.e. "the same value of `courseId` does not appear in different courses").

The SSD constraints that have emerged are normally expressed using *path constraints*, logical statements whose atoms are expressions of the form $r(x, y)$ where r is a regular expression over the set Λ of labels. Intuitively, $r(x, y)$ states that y can be reached from x following the path whose labels spell a word in r. Inclusion constraints can be expressed easily as path constraints in the following way:

$$p \subseteq q \stackrel{def}{=} \forall x.p(root, x) \Rightarrow q(root, x)$$

Some integrity constraints specification formalisms are actually included in current schema languages (i.e. XML Schema), in particular for key constraints that we will discuss in the following paragraph.

A very recent proposal is [85], a pattern-based schema formalism that allows *XML structural constraint* specification. Structural constraints are based on path expressions and allow specification of path implication, path absence and path co-occurrence as basic patterns XML documents have to exhibit. In [85] is also shown how a set of structural constraint combined with a schema specification can be translated into a specialized DTD.

Key Constraints

Key constraints are used to provide a canonical identifier for a data element, are important for query optimization purposes and are used in *foreign key* integrity constraint. There are many possible generalizations of the relational notion of key to the semistructured case. Both the XML specification itself and XML Schema provide notions of key specification. In the XML standard the notion of key is provided by means of ID attributes in DTDs and it is global and unary; in XML Schema uniqueness, key, and *keyref* constraints are specified depending on XPath [1] expressions. The complexity and technicality of XPath makes reasoning about path inclusion, and hence key implication, rather difficult. A simple form of key constraints is proposed in [69] using ordinary XML attributes with a string value referred to as *single-valued attributes*. In this case an XML document tree T satisfies a key constraint on the element type τ iff for every two τ nodes in T, if they agree on the values of the key attributes, then they are the same node. A foreign key can be simply specified with an inclusion constraint combined with a key.

Another notion of key, useful to capture the semantics of IDREFS attributes in DTDs, is the *set-valued* foreign key on single attributes. A set-valued foreign key constraint $fk_S(\tau_1.a_1, \tau_1.a_2)$ (with τ_1 and τ_2 element types, a_2 a single-valued

attribute of τ_2, and a_1 a set-valued attribute) asserts that for any τ_1 node x and any value v in the a_1 attribute of x, there exists a τ_2 node y such that v matches the value of the $y.a_2$ attribute. Roughly following the notion of XML Schema keys, key specifications based on path expressions have also been studied in [20], introducing two notions of keys:

Absolute key it is a constraint of the form: $(Q, \{P_1, \ldots, P_l\})$ where Q is a path expression called the *target path* and $\{P_1, \ldots, P_l\}$ is a set of path expressions called the *key paths*. In an XML tree, Q identifies a set of nodes on which the key is defined, denoted by $[\![Q]\!]^{\mathbf{t}}_{;}$. The idea is that, as for key attributes in relational databases, the key paths emanated from nodes in $[\![Q]\!]^{\mathbf{t}}_{;}$ provide an identification of nodes in $[\![Q]\!]^{\mathbf{t}}_{;}$. A tree satisfies the key iff for every two nodes n_1, n_2 in $[\![Q]\!]^{\mathbf{t}}_{;}$, if they have all the key paths and agree on them (meaning that walking key paths we obtain the same values), then they are the same node.

Relative key in many scientific data formats there is a hierarchical key structure in which subelements are located relative to some parent node. To describe this we can use the relative key $(R, (Q, S))$ where R is a path expression that identifies a set of nodes $[\![R]\!]^{\mathbf{t}}_{;}$ (the "parent" entities), and (Q, S) is a key for every "sub-document" rooted at a node in $[\![R]\!]^{\mathbf{t}}_{;}$.

Note that these key notions capture the semistructured nature of XML by allowing the absence or the presence of more than one key in some paths, so both absolute and relative keys are *optional* and can be *duplicated*. In Section 9.2.2 we show how these concepts can be expressed using a spatial tree logic.

8.2.2 Query languages

Many query languages for semistructured data and XML have been designed in the past years: StruQL, Lorel, XQL, XML-QL, YATL, etc. Building on this research, W3C is designing XQuery [48], a standard query language for XML data, which subsumes many concepts coming from these languages. XQuery (still a work in progress) is a typed, Turing-complete query language that can be used for XML-enabled database systems and native XML systems. The query paradigm of XQuery consists of two parts: (i) a *pattern* is used to extract bindings for a set of variables, and (ii) a *construct* clause indicates how to build the answer from the set of bindings found in (i). The pattern in (i) is inspired by conjunctive queries with navigational facilities provided by path expressions.

XSLT (W3C Web site) is an alternative to *pattern-construct* paradigm languages: it allows definition of *tree transformations* (that would require the knowledge of the document structure in XQuery-style languages) using structural recursion on trees.

Another interesting different approach for XML processing is examined in [75], where a *regular expression pattern matching* is proposed. Regular expression pattern matching is similar in the spirit to the pattern matching facilities found in languages

in the ML family. Its extra expressiveness comes from the use of regular expression types to dynamically match values. The main difference w.r.t. *pattern-construct* query languages is the "single-match" semantics, i.e. there is only one binding for a given pattern-match (in contrast to "all-matches" set of bindings of XQuery and TQL). In [75] a "first-match" policy is also investigated for ambiguous repetition and alternation patterns.

TQL

TQL is a query language for semi-structured data based on a spatial logic for trees. We give here only a brief presentation of the language; for a complete formal exposition see [42], for an informal one see [58].

Consider the following bibliography, where, informally, a[F] represents a piece of data labeled a with contents F (the data model is the same of our logic, and will be fully defined in the Section 9.1, we remark that it is unordered); F is empty, or is a collection of similar pieces of data, separated by "|". When F is empty, we can omit the brackets, so that, for example, Darwen[ ] can be written as Darwen.

The bibliography below consists of a set of references all labeled book. Each entry contains a number of author fields, a title field, and possibly other fields.

$BOOKS =$ book[author[Date] | title[DB] | publisher[Addison-Wesley]] |
book[author[Date] | author[Darwen] | title[Foundation for Future DB]
| year[2000] | pages[608]] |
book[author[Abiteboul] | author[Hull] | author[Vianu]
| title[Foundation of DB] | publisher[Addison-Wesley] | year[1994]]

Suppose we want to find all the books in *BOOKS* where one author is Date; then we can write the following query (hereafter X, x, and names beginning with $ denote variables, everything else is a constant):

from $BOOKS \vDash$ .book[X], X $\vDash$ .author[Date] *select* text[X]

The query consists of a list of *matching expressions* contained between from and select, and a *reconstruction expression*, following select. The matching expressions bind X with every piece of data that is reachable from the root *BOOKS* through a book path, and such that a path author goes from X to Date; the answer is $text[author[Date] \mid title[DB] \mid \ldots] \mid text[author[Date] \mid author[Darwen] \mid \ldots]$, i.e. the first two books in the database, with the outer *book* rewritten as *text*. The operator .book[X] is actually an abbreviation for book[X] | $\mathbf{T}$. The $BOOKS \vDash$ book[X] | $\mathbf{T}$ statement means: *BOOKS* can be split in two parts, one that satisfies book[X], the other one that satisfies $\mathbf{T}$. Every piece of data satisfies $\mathbf{T}$ (*True*), while only an element book[...] satisfies book[X]; hence, $BOOKS \vDash$ book[X] | $\mathbf{T}$ means: "there is an element book[X] at the top level of *BOOKS*".

In TQL, a matching expression is actually a logic expression, combining matching-like and classical logical operators. For example, the following query combines path-expression-like logical operators and classical logical operators ($\forall$, $\Rightarrow$) to get schema information out of the data source. It retrieves the tags appearing into each book.

$$\textit{from} \quad \textit{BOOKS} \vDash \forall \mathrm{X}(.\mathrm{book}[\mathrm{X}] \Rightarrow .\mathrm{book}[\mathrm{X} \wedge .\mathrm{x}[\mathbf{T}]]) \quad \textit{select} \quad \mathrm{tag}[\mathrm{x}]$$

The query can be read as: get tag[x] for those labels x such that, for each book book[X], x is the tag of one of the elements of the book. Observe how the free variable x carries information from the binder to the result. The same property is expressed below using negation, as 'there exists no book where x is not a sub-tag.

$$\textit{from} \quad \textit{BOOKS} \vDash \neg\ .\mathrm{book}[\neg\ .x[\mathbf{T}]] \quad \textit{select} \quad \mathrm{tag}[x]$$

For more examples, see [42, 58]. In particular in [58] [2] is shown how TQL can be used to express and check semi-structured data properties (such as types and key constraints) and to retrieve all pieces of data that satisfy these properties.

The current TQL system is available at `http://tql.di.unipi.it/tql` and it is based on a TQL Algebra [54, 55] dealing with possibly infinite tables. However the work on design and implementation of TQL is far from finished. At language level the system could be integrated with a static analysis based on the expected results of this Thesis (e.g. static declaration of empty results and logical rewritings).

At the implementation level, we are working toward the design of better persistent data structures and physical operators, endowed with a cost model, to allow cost-based physical optimization. This work is still preliminary, though, since we have no usage model of the language to guide us.

Comparing TQL with XQuery

While TQL and XQuery are based on the same *pattern-construct* paradigm, they differ in many aspects.

First of all, TQL, by design, is based on a logic that can express types, constraints, and queries, and is tailored for formal, and automated, manipulation. On the other side, XQuery is designed as an industrial-strength language, aimed at both database-oriented and document-oriented applications. As a consequence, TQL has a very sharp semantic definition, that can be completely defined in one page of formulas, while XQuery semantics is much more complex. On the other side, XQuery data model supports order and *oid-like* information, which are not dealt with in the current TQL version.

Second, even though XQuery expressive power is greater than TQL's (the former is Turing-complete), some queries can be more easily expressed in TQL, thanks

[2] In [58] the TQL syntax used is the concrete syntax of the current implementation and differs in some details (i.e. keywords and PC_DATA extension) from the abstract syntax shown here.

to the greater expressive power of the tree-logic with respect to a pure matching mechanism.

Finally, XQuery features powerful vertical navigational facilities, while it lacks corresponding horizontal operators; TQL, instead, makes no difference between horizontal and vertical navigation, hence allowing the user to easily impose horizontal constraints on documents.

8.2.3 Reasoning, rewriting and query optimization

In relational databases, we are often interested in knowing whether a given set of dependencies can be satisfied. In addition, if an instance satisfies a set of dependencies, it is useful to know which other dependencies are necessarily satisfied by that instance (implication problem). These problems can also be defined in the context of constraints for SSD and XML, i.e. for path and SSD key constraints. In particular the *implication problem* for path constraints is an important issue. Results on decidability and complexity of the general implication problem and implication problem for path constraints are provided in [7].

Some extension of the implication problem and the interaction of schema and constraints are studied in [24, 25], where it is shown that schemas have a significant impact on the constraint implication problem: some instances of the problem that are decidable in the schema-less case become undecidable when schema is present, and conversely.

In [21] satisfiability and logical implication problems are studied for several key constraint languages (in the form of absolute key constraint), providing efficient reasoning for some limited languages. Satisfaction and implication for XML keys is investigated in [10].

Another approach to represent and reason about the structure of documents (schema and constraints) is provided in [36] and it is based on Description Logics.

Optimization

The main problem in query optimization for SSD and XML is the lack of a well accepted framework for storing and accessing XML documents. This reduces any *physical* optimization approach to specific cases where some assumptions are made about how XML documents are stored and accessed. This problem should be alleviated in presence of a standard intermediate Algebra for XML, independent from the physical layer. However many optimizations based only on logical rewritings are possible, i.e. rewriting based on known constraints and schema. Schema-based optimization schemes for general types of path queries are discussed in [25] in the context of UnQL query language and algebra. But the underlying schema formalism of that proposal is weaker than DTDs. Recently, in [84] and in [9] other optimization techniques for schema-based path expression have been proposed. These techniques are based on the notion of path equivalence classes (PECs) on a schema. Using

PECs it is possible to determine redundant path conditions, path shortenings and unsatisfiable paths at compile time. Reasoning on key and other integrity constraints can also be used to simplify queries and path expressions (i.e. [7]). In any case, up to now, there is no proposal of query optimization techniques or rewriting systems considering both schema and constraints implications.

Chapter 9

A Query Language for Web Data

In this Chapter we report some examples and intermediate results developed in the early stage of this Thesis. We were working on the TQL language as a unifying framework to query, describe and reason about semistructured data. In subsequent work we preferred BiLog as the real unifying framework, it could be interesting to develop a query language inspired by BiLog and continue in this direction.

We start from the TQL logic, we define some useful derived connectives, and we perform a "tree logic translation" of the usual schema and constraint specifications for SSD. This is partially done in our previous work [58] as an informal presentation of the TQL expressive power. Here we reformulate this from the logic point of view, introducing the notion of *constraint and type specifications* as relations between a data source and a closed spatial tree logic formula (essentially, a satisfaction relation).

The following step is to use these constraint and types specifications to reason about constraints and types, that is to solve decision problems such as: schema inclusion, constraint consistency, and constraint implication in presence of a determined type. Reasoning about constraints and types corresponds, in our context, to a validity/satisfiability check of a determined formula. Thus, deciding validity in a reasonable time is the crucial problem. In particular we are interested in addressing which TQL sub-logic has a validity algorithm.

In addition we can use constraints and types (that are known to be valid) in order to rewrite formulas and optimize or error-check TQL queries. Here we give only an idea of the various optimization and static analysis techniques that arise using constraints.

9.1 TQL Logic Presentation

TQL logic is a subset of the modal ambient logic for trees [43] enriched with tree variables, and recursion. Formal syntax and semantics of the logic are studied and presented in [42]. Here we briefly present the language that we will use to specify

semistructured data properties and binding expressions.

9.1.1 TQL formulas

The following table reports the primitive TQL logic connectives. The symbol $\sim$ denotes a binary operator belonging to a fixed set of label comparison operators, such as $=$, $\leq$, closed under negation. There are three types of variables:

- the tree variable $\mathcal{X}$, hereafter denoted also by strings with prefix \$ followed by an upper case letter;
- the label variable x, hereafter denoted also by strings with prefix \$ followed by a lower case letter;
- the recursive variable ξ.

In a formula A, variables that are not bound by $\exists x$, $\exists X$, or $\mu\xi$, are *free* in A. In the syntax below, we write E_v whenever the variable v is bound in the scope E, as in $\exists X.A_X$.

Table 9.1.1. *Primitive Logical Formulas:*

$\eta ::=$	label expression
n	label constant ($n \in \Lambda$)
x	label variable
$A, B ::=$	formula
$\mathbf{0}$	empty tree
$\eta[A]$	location
$A \mid B$	composition
$\mathbf{T}$	anything
$\neg A$	negation
$A \wedge B$	conjunction
$\mathcal{X}$	tree variable
$\exists x.A_x$	quantification over label variables
$\exists X.A_X$	quantification over tree variables
$\eta \sim \eta'$	label comparison
ξ	recursion variable
$\mu\xi.A_\xi$	recursive formula (least fixpoint); ξ may appear only positively

Essentially, this logic extends STL with variables, quantification, label comparison, and recursion. We informally describe the new connectives introducing additional properties of the satisfaction relation $\models FA$.

comparison:	$n \sim m$	implies	$\models Fn \sim m$
$\exists$ label :	$(\exists n.\ \models FA\{x \leftarrow n\})$	implies	$\models F\exists x.A$
$\exists$ tree :	$(\exists F'.\ \models FA\{X \leftarrow F'\})$	implies	$\models F\exists \mathcal{X}.A$
fix point:	if F is contained in the least fixpoint of the function $\lambda\xi.A$ taken over the collection of sets of labeled trees ordered by inclusion, then $\models F\mu\xi.A$		

9.1.2 Derived connectives

As usual, negation allows us to derive useful 'dual' logical operators. In the following tables we extend the language with these new operators and we define them in term of base ones.

Table 9.1.2. *Dual connectives:*

$A, B ::=$		formula (we list here the dual ones)
$\eta[\Rightarrow A]$	$\stackrel{def}{=}\ \neg(\eta[\neg A])$	if-location
$A \mid\mid B$	$\stackrel{def}{=}\ \neg(\neg A \mid \neg B)$	decomposition
$\mathbf{F}$	$\stackrel{def}{=}\ \neg\mathbf{T}$	false
$A \vee B$	$\stackrel{def}{=}\ \neg(\neg A \wedge \neg B)$	disjunction
$\forall x.A$	$\stackrel{def}{=}\ \neg(\exists x.\neg A)$	quantification over label variables
$\forall X.A$	$\stackrel{def}{=}\ \neg(\exists X.\neg A)$	quantification over tree variables
$\nu\xi.A$	$\stackrel{def}{=}\ \neg(\mu\xi.\neg A\{\xi \leftarrow \neg\xi\})$	greatest fixpoint; ξ may appear only positively

To appreciate the difference between $|$, $m[A]$ and $||$, $m[\Rightarrow A]$ operators, consider the following statements.

- There exists a decomposition of F into F' and F'', such that F' satisfies $book[A]$, and there is no constraint upon F''; i.e., there is a book inside F that satisfies A: $I \models book[A] \mid \mathbf{T}$;

- for every decomposition of F into F' and F'', either F' satisfies $book[\Rightarrow A]$ or F'' satisfies $\mathbf{F}$; i.e., every book inside I satisfies A: $I \models book[\Rightarrow A] \mid\mid \mathbf{F}$.

9.1.3 Path formulas

Most query languages for semistructured data use *regular path expressions* [48, 53, 68] to retrieve information found at the end of any path described by a regular expression over labels. In addition path expressions and path languages are widely used to specify constraints over SSD and XML [23, 20, 21, 24].

TQL Logic is expressive enough to model regular path expressions by defining *path formulas* as derived operators that allow the programmer to describe the set of paths to be followed, and to bind variables to what is found at the end, or in the

middle, of the path. We can define the set of the paths of a tree as a language over the alphabet Λ, as follows: the sets of paths of $\mathbf{0}$ only contains the empty string, the one of $m[P]$ contains a path $m.p$ for any path p in P, and the set of paths of $P \mid P'$ is the union of the sets of paths of its two components; e.g., $a.b$ and $a.c$ are the only paths of both $a[b[\,] \mid c[\,]]$ and $a[b[\,]] \mid a[c[\,]]$.

Table 9.1.3. *Path formulas*

$\alpha ::=$	label matching expression
n	matches n
$\neg\,\alpha$	matches whatever α does not match
$\beta ::=$	path element
$.\,\alpha$	some edge matches α
$!\alpha$	each edge matches α
$p, q ::=$	path
β	elementary path
pq	path concatenation
$p*$	Kleene star
$p \vee q$	disjunction
$p(X)$	naming the tree at the end of the path

Considering the following statement: X is some book found in \$BOOKS collection, and the only author of X is Abiteboul. We can express it using the syntax of path expressions as:

$$\$BOOKS \models .book(X)!author[Abiteboul]$$

The semantics of path formulas is defined by a translation $[\![A]\!]$ that maps them into the base operators. This interpretation translates all non-path operators as themselves (e.g., $[\![A \mid A']\!] = [\![A]\!] \mid [\![A']\!]$), and translates path operators as specified below. For example, we have:

$$\begin{aligned}&[\![.book(X)!author[Abiteboul]]\!]\\ =\;& \exists x.\ x = book \wedge x[X \wedge (\forall x'.\ x' = author \Rightarrow x'[\Rightarrow Abiteboul] \,||\, \mathbf{F})] \mid \mathbf{T}\end{aligned}$$

Table 9.1.4. *Translation of path formulas:*

$Matches(x, \eta)$	$\stackrel{def}{=}$	$x = \eta$
$Matches(x, \neg\alpha)$	$\stackrel{def}{=}$	$\neg Matches(x, \alpha)$
$[\![\,.\alpha[A]]\!]$	$\stackrel{def}{=}$	$(\exists\, x.Matches(x, \alpha) \ \wedge\ x[[\![A]\!]]) \mid \mathbf{T}$
$[\![!\alpha[A]]\!]$	$\stackrel{def}{=}$	$(\forall\, x.Matches(x, \alpha) \ \Rightarrow\ x[\Rightarrow [\![A]\!]]) \,||\, \mathbf{F}$
$[\![pq[A]]\!]$	$=_{def}$	$[\![p[[\![q[A]]\!]]]\!]$
$[\![p * [A]]\!]$	$=_{def}$	$\mu\xi.[\![A]\!] \ \vee\ [\![p[\xi]]\!]$
$[\![(p \vee q)[A]]\!]$	$=_{def}$	$[\![p[A]]\!] \ \vee\ [\![q[A]]\!]$
$[\![p(X)[A]]\!]$	$=_{def}$	$[\![p[X \ \wedge\ A]]\!]$

9.2 Expressivity

TQL formulas describe properties of information trees (an unordered tree model for semistructured data and XML). Schema, types and constraints of SSD are, essentially, properties that a given data source must satisfy. More generally, given a closed TQL formula A and an information tree F, every statement of the form $\models FA$ can be viewed as a constraint on the data source represented by F. In what follows we call *constraint/type formula* a generic closed formula, and *constraint/type specification* the satisfaction statement of a constraint/type formula w.r.t. a data source F. We specify constraint specification also in the form $\models \mathcal{X}A$ meaning that the constraint formula A is satisfied by the information tree bound to the variable X.

In this section we show some constraint formulas that declaratively impose structure to data (types) or constraint values (traditional integrity constraint), all these formulas can be simply plugged inside a TQL *from-select* clause to validate the current status of the data. Further investigation on the expressive power is planned, in particular a detailed comparison with tree automata.

9.2.1 Expressing Schema and Types

Traditional path-based query languages explore the vertical structure of trees. Our logic can also easily express horizontal structure, as is common in schema for semistructured data. (E.g. in XML DTDs, XDuce Types [76], and XSD Schema [2]; however, the present version of our logic only considers unordered structures).

Regular Expression Types

To express constraints for regular expression types, we can extract the following regular-expression-like sublanguage, inspired by XDuce and XSD types. Every expression of this language denotes a set of information trees:

$\mathbf{0}$	the empty tree
$'\% \stackrel{def}{=} \exists x.\ x[\mathbf{0}]$	a basic value leaf
$A \mid B$	an A next to a B
$A \vee B$	either an A or a B
$n[A]$	an edge n leading to an A
$\%[A] \stackrel{def}{=} \exists x.\ x[A]$	whatever edge leading to an A
$A^* \stackrel{def}{=} \mu\xi.\ \mathbf{0} \vee (A \mid \xi)$	a finite multiset of zero or more A's
$A^+ \stackrel{def}{=} A \mid A^*$	a finite multiset of one or more A's
$A? \stackrel{def}{=} \mathbf{0} \vee A$	optionally an A
$\mathbf{T}$	anything

This formulas can be used to express non-recursive types like the following, borrowed by [77]:

```
type  Addrbook = addrbook[Person*]
type  Person   = person[Name, Addr, Tel?]
type  Name     = name[String]
type  Addr     = addr[String]
type  Tel      = tel[String]
```

We express the type statement `\$Abook: Addrbook` as the following type constraint specification:

$$\$Abook| = addrbook[person[name['\%]|addr['\%]|tel['\%]?]*]$$

Recursive types can be translated in our logic using the modal recursion operator $\mu\xi.A$. For example, the following recursive type

```
type Section = section[intro[String], Section*, conc[String]]
```

representing the DTD in Section 8.2.1 can be expressed with the following formula:

$$section[\mu\xi.intro['\%] \mid section[\xi]^* \mid conc['\%]]$$

We remark that the data model is unordered so this constraint does not imply any tag precedence. For more complex recursive types involving several type definitions we can use a recursive variable for each type, starting the translation from the root. For example the type:

```
type Part = part[name[String], Subpart*]
type Subpart = subpart[material[String], Part*, Subpart*]
```

with `Part` as root, can be represented with the formula:

$$\mu\xi_{Part}.\ part[name['\%] \mid \mu\xi_{Subpart}.\ subpart[material['\%] \mid \xi^*_{Part} \mid \xi^*_{Subpart}]]$$

We are actually still investigating on the expressive power of our logic, but we claim that we can express XML Schema, up to document order and XML attributes.

Types for complex structures

The use of quantification, negation, recursion and horizontal composition allows us to express complex kinds of types in terms of structure of forests. The Tree logic can be used easily to model and express structural property of forest. A simple property is the tree shape of a forest that can be expressed as: $\exists x.x[\mathbf{T}]$. However this property can be expressed also with the regular-expression-type $\%[\mathbf{T}]$. By negation of this formula we obtain the class of non-tree forests (i.e. $\neg\exists x.x[T]$).
Here we report some complex types that we can express easily:

- unary tree: $\mu\xi.\ \mathbf{0}\ \vee\ \%[\ \xi\ ]$
- binary tree: $\mu\xi.\ \mathbf{0}\ \vee\ \%[\ \xi\ ] \mid \%[\ \xi\ ]$
- forest with an odd number of branches: $\mu\xi.\ \%[\mathbf{T}] \vee (\%[\mathbf{T}] \mid \%[\mathbf{T}] \mid \xi)$
- a collection of trees with the same label: $\exists x.(x[\mathbf{T}])^*$

9.2.2 Expressing Constraints

While types constrain the shape of data, it is often useful to constrain the values as well; the canonical examples are key constraints and referential integrity constraints. To exemplify these notions in our logic we borrow from [23] the typical student-course use case.

$$\begin{array}{l}\$S = student[SSN['\%] \mid name['\%] \mid taking['\%]^*] \ in \\ \$C = course[cno['\%] \mid title['\%] \mid taken\_by['\%]^*] \ in \\ \models \$CS\$S^* \mid \$C^*\end{array}$$

The stored relation is redundant, but this is useful to model the notion of inverse constraint.

Inclusion and Inverse Constraints

Inclusion constraint are extent constraint that can be expressed by path containment. In our logic we express them using path, implication, and quantification connectives:

$$\begin{array}{l}\models \$CS\forall\mathcal{X}.\ .student.taking[\mathcal{X}] \Rightarrow .course.cno[\mathcal{X}] \\ \models \$CS\forall\mathcal{X}.\ .course.taken\_by[\mathcal{X}] \Rightarrow .student.SSN[\mathcal{X}]\end{array}$$

Here we have formulated these constraints knowing the schema. If we want *global* constraints, that is constraints working over a document whose structure is not known or is defined by a DTD, we specify:

$$\begin{array}{l}\models \$CS\forall\mathcal{X}.\ .\%* .student.taking[\mathcal{X}] \Rightarrow .\%* .course.cno[\mathcal{X}] \\ \models \$CS\forall\mathcal{X}.\ .\%* .course.taken\_by[\mathcal{X}] \Rightarrow .\%* .student.SSN[\mathcal{X}]\end{array}$$

In this case we constrain every taken course of a student reachable from the root to be actually a course reachable from the root (and conversely).

Here there is an example of inverse constraint, that is constraint expressing that two relationship are symmetric.

$$\forall\$C.\forall\$S.\ .student[.SSN[\$S] \wedge .taking[\$C]] \Leftrightarrow .course[.cno[\$C] \wedge .taken\_by[\$S]]$$

Key Constraint

As done by Buneman et al. [20], we can express a notion of *relative* keys. Assume you have a set of books whose type, expressed as in the previous section, is:

$$\$BOOKS \vDash books[\ book[\ chapter[number[\mathbf{T}] \mid content[\mathbf{T}]]* \ ]* \ ]$$

we say that *number* is a key for *chapter* relative to *books.book*, and this means that, for each specific book, it is never the case that two different chapters have the same number. Of course, *number* is not an *absolute* key for *books.book.chapter*, since two

different chapters may have the same number, if they belong to two different books. This is expressed in TQL by the following formula.

$$\$BOOKS \vDash \neg books.book[\,.chapter.number[X] \mid .chapter.number[X]\,]$$

A positive version of the formula can be used to find any chapter number that violates the constraint, and the involved book Y:

$$\begin{array}{ll} from & \$BOOKS \vDash books.book(Y)[\,.chapter.number[X] \mid .chapter.number[X]\,] \\ select & ReusedChapterNumbers[\,book[Y] \mid number[X]\,] \end{array}$$

This key notion alone (with no schema definition) does not constrain the key to appear in each element. A notion of mandatory key can be expressed in our logic by adding constraints of the form $\neg\ .books.book.chapter[\neg\ .number[\mathbf{T}]]$, that can be expressed using derived path connectives as $!books!book!chapter.number[\mathbf{T}]$.

The notion defined in [20] is slightly more complex. The relative key constraint we have shown is there described as $(books.book(chapter,(number)))$, which is a special case of a more general constraint $(Q,(Q',(P_1,\ldots,P_n)))$.

$(Q,(Q',(P_1,\ldots,P_n)))$ specifies that, for each element e that can be reached through the path Q from the root (each book) and for each two different sub-elements e', e'', reachable from e through Q' (e.g., two chapters of the same book) one key-path P_i exists such that any sub-element of e' reachable through P_i is different from any sub-element of e'' reachable through P_i. This is quite long to express in first-order logic, and we even cheated a little, since the actual definition distinguishes node-equality, used to compare e' and e'', from value-equality, used to compare their P_i-reachable sub-elements (see [20]).

In our logic, the same notion can be expressed, without cheating, as:

$$\forall X_1 \ldots \forall X_n.\ \neg.Q[\ .Q'[.P_1[X_1] \wedge \ldots \wedge .P_n[X_n]] \mid .Q'[.P_1[X_1] \wedge \ldots \wedge .P_n[X_n]]\ ]$$

Foreign key constraints can be expressed in terms of key constraints combined with inclusion constraints. As an example, consider the following schema, describing a list of books and a list of authors.

$$\begin{array}{l} books[\ book[\ author[auth\text{-}id[\mathbf{T}]]* \mid \mathbf{T}\ ]*\ ] \\ \mid authors[\ author[\ id[\mathbf{T}] \mid \mathbf{T}\ ]*\ ] \end{array}$$

The foreign key constraint specifies that: (i) each author is identified by an *auth-id*, a classic key constraint $KEY(auth\text{-}id)$; (ii) the referential integrity constraint, the *auth-id*'s have to be included into the actual *id*'s of registered authors

$$KEY(auth\text{-}id) \wedge \forall X.\ .books.book.author.auth\text{-}id[X]\ \Rightarrow\ .authors.author.id[X]$$

As a conclusion, our logic allows types and constraints to be easily specified, and with our logic we can express new notions of key, such as *mandatory keys* or *not duplicated keys*.

9.3 Reasoning and Optimization

In TQL logic the satisfaction problem $\models FA$ is decidable. Indeed, the TQL system solves the more general problem of *query answering*, that is an extended satisfaction $\models FA$ collecting all the pieces of data F' that substituted to the free (label or tree) variables X of A satisfy $\models FA\{X \mapsto F'\}$.

The satisfaction of spatial formulas in not always decidable, e.g. the satisfaction problem for the Ambient Logic with the guarantee operator $(A \triangleright B)$ is undecidable. However in many interesting sub-logics of the Ambient Logic, the problem of whether $\models FA$ becomes decidable [43]; some complexity results are also known [49].

Most decision problems over SSD types, constraints and queries can be rephrased as validity (or satisfiability) of tree logic formulas (see Section 8.1), thus validity and satisfiability algorithms for TQL sub-logics can be used to solve decision problems over SSD represented as information trees.

In the full TQL logic (including quantification and recursion) the *validity problem* is, in general, undecidable. But in [32] is shown that in the following logic the validity problem is decidable (and equivalent to the satisfaction problem):

Table 9.3.1. *Primitive Ground Formulas:*

$A, B ::=$	formula
$\mathbf{F}$	false
$A \wedge B$	conjunction
$A \Rightarrow B$	implication
$\mathbf{0}$	empty tree
$n[A]$	location
$A@n$	placement
$A \mid B$	composition
$A \triangleright B$	guarantee

This logic is STL, essentially the ground logic of TQL extended with the composition and location adjunctions ($A \triangleright B$ and $A@n$).

It is clear that this language is not sufficient to express general types and constraint over SSD, however it is a good starting point to perform some preliminary reasonings. In addition a study of derived connectives of this logic can be useful to understand which TQL connectives are really needed to express standard types and constraints.

Table 9.3.2. *Derived connectives:*

$A, B ::=$		formula (we list here the dual ones)
$\neg A$	$\stackrel{def}{=} A \Rightarrow \mathbf{F}$	negation
$\mathbf{T}$	$\stackrel{def}{=} \neg\mathbf{F}$	true

$A \vee B$	$\stackrel{def}{=}$	$\neg(\neg A \wedge \neg B)$	disjunction
$A \Leftrightarrow B$	$\stackrel{def}{=}$	$(A \Rightarrow B) \wedge (B \Rightarrow A)$	logical equivalence
$m = n$	$\stackrel{def}{=}$	$m[\mathbf{T}]@n$	label equality
$m \neq n$	$\stackrel{def}{=}$	$\neg m = n$	label inequality
$\eta[\Rightarrow A]$	$\stackrel{def}{=}$	$\neg(\eta[\neg A])$	if-location
$A \mid\mid B$	$\stackrel{def}{=}$	$\neg(\neg A \mid \neg B)$	decomposition
$A^{\exists}$	$\stackrel{def}{=}$	$A \mid \mathbf{T}$	some component
$A^{\forall}$	$\stackrel{def}{=}$	$A \mid\mid \mathbf{F}$	every component
A^p	$\stackrel{def}{=}$	$\overbrace{A \mid \ldots \mid A}^{p \text{ times}}$	p components
$Trees^{\geq p}$	$\stackrel{def}{=}$	$(\neg\mathbf{0})^p$	at least p branches
$Tree$	$\stackrel{def}{=}$	$\neg\mathbf{0} \wedge \neg Trees^{\geq 2}$	exactly 1 branch
$A^{\exists Tree}$	$\stackrel{def}{=}$	$(Tree \wedge A)^{\exists}$	some tree
$A^{\forall Tree}$	$\stackrel{def}{=}$	$(Tree \Rightarrow A)^{\forall}$	every tree

As an example, using the derived connectives defined above we can express a simplified version of non recursive types over SSD.

$\mathbf{0}$	the empty tree
$A \mid B$	an A next to a B
$A \vee B$	either an A or a B
$n[A]$	an edge n leading to an A
$A^{\bar{*}} \stackrel{def}{=} A^{\forall Tree}$	a finite multiset of zero or more A's, when $A \Rightarrow Tree$
$A^{\bar{+}} \stackrel{def}{=} A \mid A^*$	a finite multiset of one or more A's, when $A \Rightarrow Tree$
$A? \stackrel{def}{=} \mathbf{0} \vee A$	optionally an A
$\mathbf{T}$	anything

The next step is *reasoning* about constraints (and types), for example using them to optimize queries and to pinpoint that some parts of a query are not compatible with some constraint. If we focus on the TQL version of families of constraints that have already been studied, we can reuse known algorithms for this aim; for example, we can rephrase the excellent study on the manipulation of key constraint of [26] in terms of the TQL logic. Of course, the real issue is the generalization of those result, to encompass a greater subset of TQL logic. We have preliminary results on this, and we also plan to exploit the emerging results about algorithms for checking the validity of ambient logic formulas ([32]).

We can reason about types and constraints specified in this language, as an example we have the following type T:

$$addrbook[person[name[\mathbf{T}] \mid addr[\mathbf{T}] \mid tel[\mathbf{T}]?]^{\bar{*}}]$$

where the sub-formula $person[\ldots]^{\bar{*}}$ is equivalent to the regular expression $person[\ldots]^*$ because $person[\ldots] \Rightarrow Tree$; the constraint C

$$addrbook[person[\neg(name[\mathbf{T}] \mid name[\mathbf{T}]) \mid \mathbf{T}]^{\bar{*}}]$$

states that inside a *person* element the tag *name* is unique. The constraint implication $(T \Rightarrow C)$ is equivalent to the validity of the ground formula $(T \Rightarrow C)$. Using the results of [32], we can prove the validity of $(T \Rightarrow C)$ by model checking the satisfaction

$$\mathbf{0} \models \mathbf{T} \triangleright (T \Rightarrow C)$$

This example shows how tree logic can be used to reason about types and constraints, we are working on more expressive extensions of the ground logic preserving the decidability of validity (and satisfiability). On the other hand we are trying to address which logic connectives is really needed in constraint and type specification.

If we restrict ourselves to the tree logic translation of families of constraints that have been already studied, we can reuse known algorithms for deciding constraint implication; for example, we can rephrase the excellent study on the manipulation of key constraint of [21].

9.3.1 Rewritings

Complexity of TQL model checking and query execution is tricky. As an example, model checking parallel composition $(A \mid B)$ is, in general, exponential in the number of different elements of the forest. This is, essentially, due to the fact that we want check every possible *binary decomposition* of the forest F in two sub-forests F' and F'' such that $F \equiv F' \mid F''$. However equations provided by [43] can be useful in order to simplify the expressions and avoid model checking of expensive operators. The main idea is to define *rewriting rules* transforming formulas into equivalent cheaper (in model checking or query execution cost) formulas. For example we can, in most cases, rewrite $\mid$ into a much cheaper operator that only requires one to consider *elementary decompositions*, i.e. decomposition of the shape $a[P'] \mid P''$. That is because very often one argument of the $\mid$ operator expresses constraints on the decomposition that force the corresponding subtree to be a singleton $a[P']$, as happens with formulas $m[B] \mid \mathbf{T}$, in which the sub-formula $m[B]$ can only be satisfied by singleton information trees. This cheaper logical operator $|_L$ (*linear* $\mid$) is defined by:

$$A \mid_L B \stackrel{def}{=} (A \wedge \exists x.\, x[\mathbf{T}]) \mid B$$

The same approach can be extended to the dual $||$ operator.

In the TQL system we implemented a formula analysis algorithm, formalized in a type-system style, that is able to prove, in many situations, that a formula A implies $\exists x.\, x[\mathbf{T}]$, or that $(\forall x.\, x[\Rightarrow \mathbf{F}])$ implies A, hence enabling the application of the following rewriting rules.

Table 9.3.3. *Iterator linearization*

$A \mid B$	$\rightarrow$	$A \mid_L B$	if $A \Rightarrow \exists x.\, x[\mathbf{T}]$
$A \mid\mid B$	$\rightarrow$	$A \parallel_L B$	if $(\forall x.\, x[\Rightarrow \mathbf{F}]) \Rightarrow A$

Clearly such rewriting algorithm can benefit from the knowledge of types and constraints that the data source must satisfy. In addition type and constraint specification may introduce new rewritings and simplifications of the formulas, and this can be used for optimization of TQL queries in general. We are currently studying this topic, but we can give an idea of a very simple optimization; we have the following query:

$$from\ \$S| = student[\$X \wedge (SSN[\mathbf{T}] \mid \mathbf{T})] \mid \mathbf{T}\ select\ student\_with\_SSN[\$X]$$

and the systems knows that the data source $\$S$ is of the type T:

$$student[SSN['\%] \mid name['\%] \mid taking['\%]^*]^*$$

The formula $SSN[\mathbf{T}] \mid \mathbf{T}$ is implied by the type of the document, so this query is equivalent (if the type constraint holds) to:

$$from\ \$S| = student[\$X] \mid \mathbf{T}\ select\ student\_with\_SSN[\$X]$$

On the other hand queries whose formulas contradict constraints, can be statically rewritten to $\mathbf{0}$. This are probably type-error or mistake of the user, so this kind of analysis can be viewed as a weak error-check technique.

Chapter 10

Bigraphs vs XML

XML data are essentially tree-shaped resources, and have been modeled with unordered labelled trees in [37] where an important connection between semistructured data and mobile ambients was uncovered. Starting from *loc. cit.*, several works on spatial logic for semistructured data and XML have been proposed (e.g. [40, 38, 63]). Among these, a query language on semistructured data based on Ambient Logic was studied in [41] and implemented in [54, 58]. In this Chapter we enrich over such model of tree-shaped data by adding links on resource names, so as to obtain a more general model for semistructured data and XML. A similar step was taken in [39], which we improve upon by making use of the well-studied categorical structure of bigraph, which internalizes the notion of link and makes the difference between strong and structural separation explicit. In addition, bigraphs naturally model XML contexts: we thus obtain with no additional effort a logic to describe XML contexts which can be interpreted as web services or XML transformations.

Here we focus on the applications of BiLog to XML data. In particular, we first show how XML data (and, more generally, contexts or positive web services) can be interpreted as a bigraph. Equipped with such 'bigraphical' representation of XML data and contexts, we then show how different fragments of BiLog can be applied to describe and reason about XML.

The contribution of the Chapter is therefore to identify (fragments of) BiLog as a suitable formalism for semistructured data, and illustrate its expressiveness by means of selected examples.

10.1 Modeling XML Contexts as Bigraphs

The importance of the underlying hierarchical structure in XML, as well as the fact that links are used sporadically only for modeling relations between nodes, hints bigraphs as good models for XML documents. We interpret these documents as *ground bigraphs*, i.e., without either holes or inner names. The interpretation is trivial when nominal constraints are not considered (e.g. ID and IDREF attributes

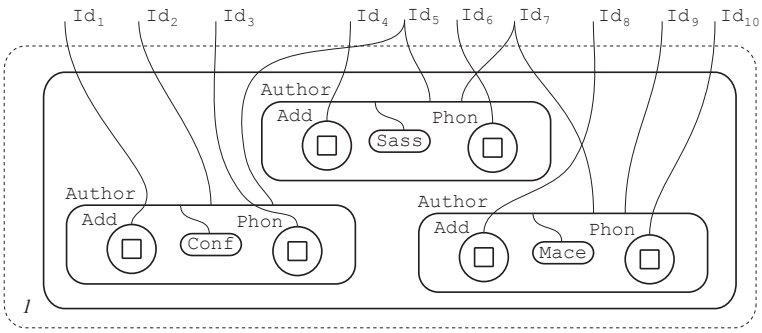

Figure 10.1: XML encoding

and namespaces), since the tree structure of XML elements is mapped into a place graph by associating controls to tags and values. In this case, there is no link between nodes, all controls have arity zero, and the XML file is completely modeled by the place graph only, in a kind of ambient like formalism [37].

When we want to model also nominal resources and links we enrich controls with identification ports and pointer ports and we connect them in the link graph. We obtain a model similar to the *trees with dangling pointers* presented in [39]. Notice that the link graph is able to model also local names (that is names protected in the model) and so also unnamed connections.

As an example, consider a database that stores scientific papers and information about their authors. We focus on the fragment quoted in the document below.

```
<authors>
 <author name="Conf" n="ID2" coauth="ID5">
   <Address n="ID1">"."</Address>   <Phon n="ID3">"."</Phon>
 </author>
 <author name="Sass" n="ID5" coauth="ID7">
   <Address n="ID4">"."</Address>   <Phon n="ID6">"."</Phon>
 </author>
 <author name="Mace" n="ID7">
   <Address n="ID8">"."</Address>   <Phon n="ID10">"."</Phon>
 </author>
</authors>
```

Tag *Author* has an identifier, $\texttt{ID}_i$, a link to another author, `coauth`, that is an `IDREF` attribute, and a general attribute, `name`. In the corresponding bigraphical encoding, see Fig. 10.1, every tag *Author* is associated to a control of arity three. Exploiting the order of the ports, we identify a port with the corresponding XML attribute unambiguously. In the picture we assume the ports ordered clockwise. The first port corresponds to the general attribute `name`, and is connected by a close link (an edge) to a value. The second one corresponds to the identifier, `ID`, and

is connected to an outer name. The final attribute corresponds to the reference, `coauth`, and is connected to a name that correspond to another *Author* tag.

The parent-child relationship on nodes in bigraphs does not capture order among children of the same node. So bigraphs can be seen as a (ordered) list of unordered (contexts of) trees connected through links. This model can be used for XML data whose document order is not relevant. Such a document arises, for instance, in XML encodings of relational databases, in the integration of semi-structured database sources, or in the case of distributed XML documents in a P2P environment.

More generally, a bigraph can be seen as a context for unordered XML data, just because there can be holes in it. So in the previous example we can imagine to put holes in place of some nodes. This yields a context that can be interpreted as a contextual XML document, that is function (*web service*) taking a list of XML files and returning their composition in context, by fitting every file in the relative position (as marked by its position on the list). In this way we can model web services, besides plain XML documents.

10.2 BiLog for XML Contexts

In [61] we introduced BiLog for bigraphical structures. In 10.1 we have shown that XML (unordered) data and contexts can be modeled as bigraphical structures. This section briefly introduces BiLog, and explains how it can be used for describing, querying and reasoning about XML. In particular, we analyze three possible cases: (i) logics for place graphs to model XML data trees and tree contexts (without considering nominal resources); (ii) logics for *discrete bigraphs* (essentially trees with unique identifiers) for XML with identified nodes; (iii) bigraphical logics for XML with soft-link connections (implemented with nominal resources, eg. ID-IDREF pointers or namespaces).

XML without `IDs`

As mentioned previously, without nominal resources XML amounts to unordered labelled tree. In [37] the author shows that such a model has some similarities with ambient calculus terms, which [41] uses to introduce a query language for semistructured data based on Ambient Logic. In [61] we show that the static fragment of ambient logic (STL) can be easily extended to the Place Graph Logic (PGL in the following) to model general contexts of tree-shaped resources. In particular PGL can describe place graphs, that is bigraphs without links, and so it can be used to talk about XML contexts (without attributes) using the encoding we defined in the previous section. We briefly present here some operators of PGL, and we informally show their semantics in the XML case. (Some of the connectives are derived; the reader is referred to [61] for the details.)

Table 10.2.1. *PGL: Place Graph Logic (some operators)*

$A, B ::=$	formulas
$\mathbf{F}$	false
$A \Rightarrow B$	implication
$\mathbf{1}$	empty single rooted bigraph
$\mathbf{id}_n$	identity on n number of holes (even zero)
$A \otimes B$	decomposing in two place graphs one next to the other
$A \circ B$	decomposing in two place graphs one inside the other
$A \circ\!\!- B$	if inserted in a context A (with s holes) then B
$\mathsf{K}(A)$	a control K containing something satisfying A
$A \mid B$	decomposing in two trees whose merge is the current model

The formulas $\mathbf{F}$ and $A \Rightarrow B$ are standard and the other propositional connectives $\mathbf{T}$, $\neg, \wedge, \vee, \Leftrightarrow$ are derived as usual. There are spatial constants $\mathbf{1}$, **join**, and id_n denoting a singleton place graph (interpreted as a single XML context). We interpret $\mathbf{1}$ to be the empty XML context, **join** the context merging two XML contexts in one, while **id** is the identity context, which transforms XML trees to themselves. The two spatial operators $A \otimes B$ and $A \circ B$ express two ways of composing contexts. The first is *horizontal* and produces a (ordered and separated) pair of contexts one next to the other. The second one is *vertical* and corresponds to fill the s holes of a context satisfying A with the context satisfying B. They are both non commutative. From this operators we can derive the Ambient-like operators for trees: $\mathsf{K}(A)$ is the context that inserts a new root labelled K in the top of a single XML context satisfied by A, and $A \mid B$ (parallel composition) denotes contexts obtained by merging the tree contexts satisfying A and B in a single root. Note that, since parallel composition performs a merge of the contexts, it provides a commutative monoid with $\mathbf{1}$ as neutral element. An interesting connective is $A \circ\!\!- B$, which essentially expresses that whenever the current model is inserted inside a XML context satisfying A, then the resulting context satisfies B.

In general, models of PGL are *positive* functions from m to n that given a list of m XML contexts produces a list of n XML contexts. By 'positive' we mean that they can only add structure to the parameters, and not remove or replace parts of them. In this sense, XML contexts are viewed as positive XML web services that take XML documents (possibly with calls to other web services, so that they effectively are XML contexts), and return XML documents. This is similar to the model of Positive Active XML proposed in [4], but with a remarkable difference: since our model does not handle ordered trees, we cannot restrict attention to functions between XML (active) documents. We need to use a *list* of parameters and a *list* of resulting contexts. To understand better the idea, consider the web service below.

$$wb : K_1(id_1) \mid K_2(id_2)$$

It takes two trees and puts the first inside a node labelled K_1, the second inside a

node labelled K_2, and finally produces the parallel composition of the two resulting trees. We need ordered parameters to put the right root in the right hole. A web service like this can be solely identified by a characteristic formula (corresponding to the tree), but more generally a formula like $\mathsf{K}_1(id_1) \mid \mathbf{T}$ can match all web services having at least one hole and decomposable as a node of arity one labelled K_1 in parallel to something else. In this sense a notion of *type* for web services arises. Similarly to [56], where the spatial tree logic is used to describe XML types and constraints, we can use PGL to formalize web service types and constraints.

Since also XML (active) documents are contexts, we can actually use the PGL to describe Active XML documents and web service in an unique framework. In addition, we can use an approach like TQL [41] to query Active XML documents and web service, and eventually use types to avoid web service useless invocations.

XML Contexts with identified nodes

In the previous section we focused on the tree structure only. Since logic and model have no way to directly identify resources, it is only possible to access a resource through navigation. A different approach is possible when the XML document has nominal resources, that is names identifying resources (e.g. node identifiers). In this case, the tree model can be seen as an extension of a heap memory model in which locations are referred to by names. Such names are intrinsically separated by the tensor product, which is defined only on structures with disjoint name sets. We can see such models as discrete bigraphs, i.e., place graphs with named resources but no name sharing between different resources. Since BiLog is designed as freely generated from a set of constructors, such a logic is obtained easily extending the PGL with named (identified) controls K_x and renamings $x \leftarrow y$.

The resulting logic is able to express properties of (contexts of) resources that can be accessed in two ways: as usual, by navigation through the tree structure, and by using names controls as pointers.

The logic essentially adds two operators to PGL: $\mathsf{K}_{\vec{a}}$ for named nodes and $a \leftarrow b$ for renaming these names. The control formula $\mathsf{K}_{\vec{a}}$ has a list of names, although in the case of XML with identifiers and no links only one name is needed. Thus, we write K_x to denote the node (with a hole) inside labelled K with name identifier x, and the formula K_x denotes this XML context only. The rename $a \leftarrow b$ is needed in order to map names of different sources to different identifiers (e.g., $x \leftarrow y \circ K_y = K_x$). The tensor product now constraints the models to be separated both in locality and in names, i.e., when we write $A \otimes B$ we mean that the models satisfying A and B have disjoint sets of identifiers (that is disjoint outer faces). On the other hand the composition $A \circ B$ is defined when the inner face of A and outer face of B coincide.

XML Contexts with Connections

In general XML data have connections between nodes that are not related to the parent-child relationship. These connections can be explicitly designed in a DTD (such as ID and IDREF attributes), or can also be implicit by the use of namespaces.

In order to model connections between resources and treat structures with pointers, we have to extend the model and the logic of discrete bigraph with a notion of sharing. The sharing is obtained in bigraphs through links between names of resources. In our example, we have encoded identifiers as tag names and `IDREFs` as pointers to names in the same document. In [61] we have introduced a logic for general bigraphs as a composition of a link graph logic and a place graph spatial logic. Such a combination is very expressive, and induces a hiding operator for local/private/hidden names. For the present application to XML this is only needed for the encoding of value attributes. On the other hand, we require a notion of separating conjunction with sharing, in order to express properties like: "*The author of paper X has a relationship with the author of paper Y.*" In fact, this property expresses separation on resources (different authors of different papers), but sharing on linked names. Such operator is explicitly introduced in [61] by using the tensor product of BiLog, the renaming function and the *freshness* operator of nominal logics. The main idea is that a link between names can be seen as a separation between separated names that are then linked by means of substitution.

10.3 XML Contexts encoded as Bigraphs

As proved in [94], the class of bigraphs can be axiomatised using a small set of elements. We recall the constructions below, and then relate it to XML. In our formalization, XML data are bigraphs with no holes (i.e., *ground*), while those with holes represent XML contexts.

The main constituent of a bigraph is the *discrete ion* $\mathsf{K}_{\vec{a}}$, which represents a node with one root and containing one hole. The hole can be filled with other ions in order to build a more complex tree-structure. The ion's control is K, with arity $ar(\mathsf{K}) = |\vec{a}|$, and every port of $\mathsf{K}_{\vec{a}}$ is linked to a name in the (ordered) list of names $\vec{a}$. Every name in $\vec{a}$ represent an outer name of the bigraph. Thinking in terms of XML data, a ion is seen as a *tag* with some *attributes*. Since arity is an ordinal, it is possible to identify the ports unambiguously and it is easy to associate them to attributes. We assume one designed port to be associated to a (unique) name used to identify the element, as an `ID` attribute. Other ports may be linked to other nodes' `ID` names, so acting effectively as `IDREFs`, or to internal edges connected to internal nodes, representing the general attributes of the element. Embedding a ion into the hole of another ion, represents the inclusion of the corresponding elements.

The basic place graphs are 1, id_n and *join*. Term 1 is the empty single rooted bigraph, it is the empty XML document; id_n is a context with n holes, n roots

and no internal node. It behaves like a neutral XML context if composed with a compatible XML document. Term *join* achieves the merging of two bigraphs: it consists of two holes, one root and no links. By composing merge with the product of two single-rooted XML documents we obtain a XML document whose single root has the two component documents as children.

The basic components of a link graph are $a \leftarrow b$, $a \leftleftarrows b$ and $/a$. Operator $a \leftarrow b$ represents renaming, and in the context of our interpretation of XML, it acts on `ID` attributes. Operator $a \leftleftarrows b$ associates name b to name a: when a represent an `ID` and b a `IDREF`, then $a \leftleftarrows b$ makes b a reference to a. Finally, $/a$ makes name a private, and allows nodes to be joined to one another in closed links. This operator will be used in our encoding of XML to express a link between attributes and their values.

In the presence of (unfilled) holes, terms represent *contexts* for XML data, i.e., documents with holes to be filled by XML data: composition acts by inserting documents in such holes. The tensor is defined only if the names appearing in the two components are disjoint. Therefore, any reference 'going across' must be created after the product. Note that since link graphs in general perform substitution and renaming, the outer names of g may not be outer names of $G \circ g$. This may happen either because they are renamed or because an edge has been added to the structure as effect of the composition, which makes the link *private*, i.e., without an externally-visible name.

The importance of the underlying hierarchical structure in XML, and the fact that links are used sporadically only for modeling relations between nodes, suggests the bigraphical model as a good model for XML documents. We interpret these documents as ground bigraphs by using the encoding explained below. The encoding is trivial in case the tree contains no attribute, when we can in fact easily map the tree structure of XML elements into the place graph by associating controls to tags and values. In this case, there is no link between nodes, all controls have arity zero, and the XML file is completely modeled by the place graph only (in a kind of ambient like formalism [37]).

In the case of elements with attributes, we need names to represent XML links between elements (e.g., like `ID-IDREF` relationships), and edges to represent elements' attributes. We consider the `ID`s used in XML data as names in bigraphs. The encoding is defined by assuming two functions on values:

- $K_{val}(v)$, mapping the value v to a ground bigraph corresponding to a single rooted node with no outer names, no nodes and no holes inside.

- $K_{val}(v)_a$, mapping the value v to a ground bigraph corresponding to a single rooted node with outer name a, no nodes and no holes inside.

The former function is used actually to encode values with bigraphs, the latter is auxiliary and encodes values linked to attributes.

Moreover we associate tags with ions. We assume a class $\mathcal{K}_{tag}$ of controls. We consider a tag t and first we observe that the list Att of its attributes is finite and ordered, hence we associate the list to an ordinal $\#Att$, and the elements of the list are identified by their position. Then we associate t with $K_{tag}(t, \#Att)_{\vec{u}}$, that is a ion with control $K_{tag}(t, \#Att) \in \mathcal{K}_{tag}$ and arity $\#Att$. The vector $\vec{u}$ indicates the names connected to the control; we assume the names in $\vec{u}$ to be the `ID`s associated to the attributes in Att. A value attribute is encoded as a value inside the node and connected to the port whose position marks the corresponding attribute. Identifiers (like `ID`) and links (like `IDREF`) attributes have a special interpretation. They become *names* of the tag and can be connected with other names in order to model references. As mentioned before, the connection is performed by using the link graphs constructors: $a \leftleftarrows b$, to create a reference, and $/a$, to create a closed connection for attributes. The general definition for the encoding is formalized in Table 10.3.1

Table 10.3.1. *XML documents as ground bigraphs*

$(\!\| v \|\!)$	$\stackrel{def}{=} K_{val}(v)$	value
$(\!\| v \|\!)_a$	$\stackrel{def}{=} K_{val}(v)_a$	value linked to an attribute name a
$(\!\| \vec{v} \|\!)_{\vec{b}}$	$\stackrel{def}{=} (\!\| v_1 \|\!)_{b_1} \otimes \ldots \otimes (\!\| v_n \|\!)_{b_n}$	with $\vec{v} = v_1 \ldots v_n$ and $\vec{b} = b_1 \ldots b_n$
$(\!\| \emptyset \|\!)$	$\stackrel{def}{=} 1$	empty tree
$(\!\| T \|\!)$	$\stackrel{def}{=} /\vec{a} \circ \sigma \circ K_{tag}(t, k+p+1)_{u,\vec{u},\vec{b}} \circ join_{n+k}((\!\| \vec{v} \|\!)_{\vec{b}} \otimes \alpha_1 \circ (\!\| T_1 \|\!) \otimes \ldots \otimes \alpha_n \circ (\!\| T_n \|\!))$	
with	$T = \langle t, \texttt{ID} = u, \vec{a} = \vec{u}, \vec{b} = \vec{v} \rangle\, T_1, ..., T_n\, \langle /t \rangle$	XML tree
	$\vec{a} = a_1 \ldots a_k$	link attributes
	$\vec{u} = u_1 \ldots u_k$	names
	$\vec{b} = b_1 \ldots b_p$	value attributes
	$\vec{v} = v_1 \ldots v_k$	values
	α_i	renaming the names of T_i into fresh names
	$\sigma = \alpha_1^{-1} \cup \ldots \cup \alpha_n^{-1}$	inverse renaming
	$/\vec{a} \stackrel{def}{=} /a_1 \otimes \ldots \otimes /a_p$	closure of the names in $\vec{a}$
	$join_{n+k}$	merging among $n+k$ bigraphs (definable from *join*)

In the table above, the encoding of values is simply the function $K_{val}(\,)$. The auxiliary encoding of values linked to attributes is given by $K_{val}(\,)_a$. Term 1 corresponds to the empty tree. The core of the translation is the encoding of (non empty) trees. Here, the role of *join* is to group together the (encodings of the) set of children of T and the (encodings of the) values linked to attributes. In this case, values linked to attributes are associate with a name. Observe in the encoding the use the renamings α_i to guarantee the product is defined, since it requires the names to be distinct. We choose fresh names, i.e., not appearing in T, and we obtain the renamings α_i by combining different operators such as $a \leftarrow b$. The obtained bigraph is single rooted, hence it fits in the ion associated to the tag t. After the composition with the ion,

we have to rename the names in order to formalize all the references, finally we need to close the link between the root and the evaluations of the values linked to attributes. The renaming is obtaining by considering the inverse of α_i (definable by using operators such as $a \leftarrow b$ and $a \leftleftarrows b$), and the closure is obtained by combining the closure of every name associated to an attribute.

10.4 Related Work

In [73] the relation between bigraphs and XML is studied on the other way around, i.e. XML is used to implement bigraphs and bigraphical reactive systems (while here we proposed bigraphs as a model for XML). The implementation is in a distributed Peer-2-Peer setting and has some similarities with one of the motivations that inspired us in this Thesis, a dynamic model that uses semistructured data/resources. The encoding proposed are similar in some way to ours and it could be interesting to integrate the two approaches in order to have a binary correspondence between bigraphical models and XML implementations. In this context BiLog could be easily applied to describe properties of the XML implementations and also to check whether a reduction can occur.

Chapter 11

Conclusion

A conclusion is a place where one got tired thinking.
B. F.

In this Thesis we moved a first step towards describing global resources by focusing on bigraphs. Our final objective is to design a general dynamic logic able to cope uniformly with all the models bigraphs have been proved useful for, as of today these include CCS [96], pi-calculus [80] and Petri-nets [95]. We introduced BiLog, a logic for bigraphs (and more generally for monoidal categories), with two main spatial connectives: composition and tensor product. Our main technical results are the embedding and comparison with other spatial logics previously studied.

Moreover, we have shown that BiLog is expressive enough to internalise the somewhere modality.

In particular, we have seen how the 'separation' plays in various fragments of the logic. For instance, in the case of *Place Graph Logic*, where models are bigraphs without names, the separation is purely structural and coincides with the notion of parallel composition in Spatial Tree Logic. Dually, as the models for *Link Graph Logic* are bigraphs with no locations, the separation in such a logic is disjointness of names. Finally, for *Bigraph Logic*, where models' nodes are associated with names, the separation is not only structural, but also nominal, since the constraints on composition force port identifiers to be disjoint. In this sense, it can be seen as the separation in memory structures with pointers (like the heap structure of Separation Logic).

In Section 4.5 we studied how BiLog can deal with dynamics. A natural solution is adding a temporal *next step* modality basically describing bigraphs that can compute (*react*) according to a Bigraphical Reactive System [80]. When the transparency predicate τ enables the inspection of 'dynamic' controls, BiLog is '*intensional*' in the sense of [107], namely it can observe internal structures. In the observed case, notably the bigraphical system describing CCS [96], BiLog can be so

intensional that the next step modality can be expressed directly by using the static fragment of BiLog. Notice that τ specifies which structures the logic can directly observe, while the next step modality, along with the spatial connectives, allows to deduce the structure by observing the behaviour. It would be interesting to isolate some fragments of the dynamic logic and investigate how the transparency predicate influences their expressivity and intensionality, as in [74].

The 'separation' plays differently in various fragments of the logic. For instance, in the case of *Place Graph Logic*, where the model is the class of bigraphs without names, the separation is purely structural and coincides with the notion of parallel composition in Spatial Tree Logic. The separation in the *Link Graph Logic* is disjointness of nominal resources. Finally, for *Bigraph Logic* it is a combination that can be seen as separation in a structured term with nominal resources (e.g. the trees with pointers of [33] and trees with hidden names [38]). The decidability of BiLog logics is an open question, we are working on extending the results of [32], and we are isolating decidable fragments of BiLog.

We did not introduce the existential/universal quantifiers. They are omitted as they imply an undecidable satisfaction relation (cf. [50]), while we aim at a decidable logic. As a matter of fact, we are working on extending the result of [32], and we are isolating decidable fragments of BiLog. We introduced the freshness quantifier as it is useful to express hiding and it preserves decidability in spatial logics [57].

In order to obtain a robust logical setting, we are developing a proof theory, that will be useful for comparing BiLog with other spatial logics, not only with respect to the model theory, but also from a proof theoretical point of view.

In PartIII we solved some open questions on decidability of spatial logics with protected names, some of them unexpected. An interesting direction could be to restate this theorems in the BiLog framework, providing in this way more strong results of undecidability.

We have not addressed a logic for tree with hidden names for BiLog. As a matter of fact, we have such a logic. More precisely we can encode abstract trees into bigraphs with an unique control amb with arity one. The name assigned to this control will be actually the name of the ambient. The extrusion properties and renaming of abstract trees have their correspondence in bigraphical terms by means of substitution and closure properties combined with properties of identity.

BiLog can express properties of trees with names.At the logical level we may encode operators of tree logic with hidden names as follows:

$$
\begin{array}{rcl}
\copyright\, a & \stackrel{def}{=} & ((a \leftarrow a) \otimes \mathbf{id}) \circ \mathbf{T} \\
\mathbf{C}x.\, A & \stackrel{def}{=} & (\nu x)\,(/x \otimes \mathbf{id}) \circ A \\
a \circledR A & \stackrel{def}{=} & (\neg \copyright\, a \wedge A) \vee (/a \otimes \mathbf{id}) \circ A \\
\mathbf{H}x.\, A & \stackrel{def}{=} & (\nu x)\, x \circledR A
\end{array}
$$

The operator $\textcircled{c}\, a$ says that the name a appears in the outer face of the bigraphs. The new quantifier $\mathbf{C}x.\, A$ expresses the fact that in a process satisfying A a name has been closed. The revelation $\textcircled{R}$ is a binary operator asserting the possibility of revealing a restricted name as a in order to assert A, note that the name may be hidden in the model as it has either be closed with an edge or it does not appear in the model. The hiding quantification $\mathbf{H}$ may be derived as in [44].We are currently working on the expressivity and decidability of this logical framework.

In the third part of the thesis we studied a particular resource (Web Data) and how it can be modeled with spatial logics, and with BiLog in particular. This open interesting new research directions for Web Data (possibly even active Web Data) query languages, model-checkers and validity-checkers inspired by BiLog.

Several important questions remain: as bigraphs have an interesting dynamics, specified using reactions rules, we plan to extend BiLog to such a framework. Building on the encodings of the ambient and the π calculi into bigraphical reactive systems, we expect a dynamic BiLog to be able to express both ambient logic [45] and spatial logics for π-calculus [28].

Bibliography

[1] XML path language (XPath) version 1.0 – W3C recommendation. Available at http://www.w3.org/TR/xpath.html, 2000.

[2] XML schema. Available from http://www.w3c.org, 2000.

[3] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 10 January 1999.

[4] S. Abiteboul, O. Benjelloun, and T.Milo. Positive active XML. In *Proc. of PODS*, 2004.

[5] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the WEB: From Relations to Semistructured Data and XML.* Morgan Kaufmann, San Mateo, CA, October 1999.

[6] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[7] S. Abiteboul and V. Vianu. Regular path queries with constraints. *JCSS: Journal of Computer and System Sciences*, 58, 1999.

[8] L. Acciai and M. Boreale. Xpi: a typed process calculus for xml messaging. In *Proc. of FMOODS*, 2005.

[9] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Tree pattern query minimization. *VLDB Journal: Very Large Data Bases*, 11(4):315–331, 2002.

[10] M. Arenas, W. Fan, and L. Libkin. What's hard about XML schema constraints? *Lecture Notes in Computer Science*, 2453:269, 2002.

[11] Gerard Berry and Gerard Boudol. The chemical abstract machine. In *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 81–94, New York, NY, USA, 1990. ACM Press.

[12] Biri and Galmiche. A separation logic for resource distribution: Extended abstract. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 23, 2003.

[13] Iovka Boneva and Jean-Marc Talbot. On complexity of model-checking for the tql logic. In Jean-Jacques Lévy, Ernst W. Mayr, and John C. Mitchell, editors, *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004), 22-27 August 2004, Toulouse, France*, pages 381–394. Kluwer, 2004.

[14] Iovka Boneva and Jean-Marc Talbot. Automata and logics for unranked and unordered trees. In Jürgen Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 500–515. Springer, 2005.

[15] Iovka Boneva, Jean-Marc Talbot, and Sophie Tison. Expressiveness of a spatial logic for trees. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 280–289. IEEE Computer Society, 2005.

[16] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.

[17] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. eXtensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/REC-xml, 2000.

[18] A. L. Brown, C. Laneve, and L. G. Meredith. Pi-duce: a process calculus with xml datatypes. Draft, 2004.

[19] P. Buneman. Semistructured data. In *Proceedings of the Sixteenth ACM Symposium on Principles of Database Systems*, pages 117–121. ACM Press, 1997.

[20] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. In *Proc. of International World Wide Web Conference, WWW10*, volume 39 of *Computer Networks*, pages 473–487. Elsevier, May 2001.

[21] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for XML. In *Proc. of DBPL 2001*, volume 2397 of *LNCS*, page 133. Springer-Verlag, 2002.

[22] P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data.

[23] P. Buneman, W. Fan, J. Siméon, and S. Weinstein. Constraints for semistructured data and XML. *SIGMOD Record 30*, 2001.

[24] P. Buneman, W. Fan, and S. Weinstein. Path constraints in semistructured and structured databases. In *ACM PODS*, Seattle, WA, 1998.

[25] P. Buneman, W. Fan, and S. Weinstein. Interaction between path and type constraints. In *Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems*, pages 56–67. ACM Press, 1999.

[26] P. Buneman, W. Fan, and S. Weinstein. Query optimization for semistructured data using path constraints in a deterministic data model. In *Proc. of DBPL*, 1999.

[27] R.M. Burstall. Some techniques for proving correctness of programs which alter data structures. *Machine Intellingence 7*, 1972.

[28] L. Caires and L. Cardelli. A spatial logic for concurrency (Part I). In *Proc. of Theoretical Aspects of Computer Software; 4th International Symposium, TACS 2001*, volume 2215 of *LNCS*, pages 1–37. Springer-Verlag, 2001.

[29] L. Caires and L. Cardelli. A spatial logic for concurrency (Part II). In *Proc. of CONCUR'02*, volume 2421 of *LNCS*, page 209. Springer-Verlag, 2002.

[30] L. Caires and L.Monteiro. Verifiable and executable logic specifications of concurrent objects in L_π. In *Proc. of the 7th European Symposium on Programming (ESOP'99)*, volume 1381 of *LNCS*, pages 42–56. Springer-Verlag, 2001.

[31] L. Caires and E. Lozes. Elimination of quantifiers and undecidability in spatial logics for concurrency. *Proc. of CONCUR, ext. version to appear in TCS*, 2005.

[32] C. Calcagno, L. Cardelli, and A. D. Gordon. Deciding validity in a spatial logic for trees. In *Proc. of ACM SIGPLAN Workshop on Types in Language Design and Implementation (TLDI'03)*.

[33] C. Calcagno, P. Gardner, and U. Zarfaty. Context logic and tree update. *Principles of Programming Languages 2005 (32nd POPL'2005), ACM SIGPLAN Notices*, 40(1), January 2005.

[34] C. Calcagno, P. Gardnera, and M. Hague. From separation logic to first-order logic. In *Proc. of FOSSACS*, 2005.

[35] C. Calcagno, H. Yang, and P. W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *Proc. of FSTTCS*, 2001.

[36] D. Calvanese, G. De Giacomo, and M. Lenzerini. Representing and reasoning on XML documents: A description logic approach. *JLC: Journal of Logic and Computation*, 9(3):295–318, 1999.

[37] L. Cardelli. Describing semistructured data. *SIGMOD Record, Database Principles Column*, 30(4), 2001.

[38] L. Cardelli, P. Gardner, and G. Ghelli. Manipulating trees with hidden labels. In *Proc. of Foundations of Software Science and Computation Structures (FOSSACS '03)*.

[39] L. Cardelli, P. Gardner, and G. Ghelli. Querying trees with pointers. Draft, January 2002.

[40] L. Cardelli, P. Gardner, and G. Ghelli. A spatial logic for querying graphs. In *Proc. of ICALP*, volume 2380 of *LNCS*, page 597. Springer-Verlag, 2002.

[41] L. Cardelli and G. Ghelli. TQL: a query language for semistructured data based on the ambient logic. *Mathematical Structures in Computer Science*, 14:285–327, 2004.

[42] L. Cardelli and G. Ghelli. A query language based on the ambient logic. In *Proc. of European Symposium on Programming (ESOP), Genova, Italy*, April 2001.

[43] L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proc. of POPL*. ACM Press, 2000.

[44] L. Cardelli and A. D. Gordon. Logical properties of name restriction. In *International Conference on Typed Lambda Calculi and Applications (TCLA 2001, Krakow, Poland)*, volume 2044 of *LNCS*, pages 46–60. Springer, 2001.

[45] L. Cardelli and A. D. Gordon. Ambient logic. To appear in Mathematical Structures of Computer Science, 2003.

[46] L. Cardelli and A.D. Gordon. Mobile ambients. *Theoretical Computer Science, Special Issue on Coordination*, 240(1):177–213, 2000.

[47] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Lecture Notes in Computer Science*, 1378:140–??, 1998.

[48] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery 1.0: An XML Query Language, June 2001. W3C Working Draft. http://www.w3.org/TR/xquery.

[49] W. Charatonik, S. Dal Zilio, A. D. Gordon, S. Mukhopadhyay, and J.-M. Talbot. The complexity of model checking mobile ambients. In Furio Honsell and Marino Miculan, editors, *Proc. of the 4th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2001)*, volume 2030 of *LNCS*, pages 52–167. Springer, 2001.

[50] W. Charatonik and J.-M. Talbot. The decidability of model checking mobile ambients. In *CSL: 15th Workshop on Computer Science Logic*. LNCS, Springer-Verlag, 2001.

[51] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. of the 100th Anniv. Meeting*, pages 7–18. Information Processing Society of Japan, 1994.

[52] J. Clark and M. Murata. RELAX NG. http://www.relaxng.org, 2001.

[53] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion. In *Proc. of ACM SIGMOD*, 1998.

[54] G. Conforti, O. Ferrara, and G. Ghelli. TQL Algebra and its Implementation (Extended Abstract). In *Proc. of IFIP TCS*, pages 422–434. Kluwer Academic Publishers, 2002.

[55] G. Conforti and G. Ghelli. TQL Algebra and its Implementation (Full Paper). Working draft, 2002.

[56] G. Conforti and G. Ghelli. Spatial logics to reason about semistructured data. In *Proc. of SEBD 2003: Eleventh Italian Symposium on Advanced Database Systems*. Rubettino Editore, 2003.

[57] G. Conforti and G. Ghelli. Decidability of Freshness, Undecidability of Revelation. In *Proc. of FOSSACS*, 2004.

[58] G. Conforti, G. Ghelli, A. Albano, D. Colazzo, P. Manghi, and C. Sartiani. The Query Language TQL. In *Proc. of 5th International Workshop on Web and Databases (WebDB 2002)*, 2002.

[59] G. Conforti, D. Macedonio, and V. Sassone. Bilogics: Spatial-nominal logics for bigraphs (full report). Available from `http://www.di.unipi.it/~confor/publications.html`, October 2004.

[60] G. Conforti, D. Macedonio, and V. Sassone. Bigraphical logics for XML. In *Proc. of 13 Italian Symposium on Advanced Database Systems (SEBD)*, 2005.

[61] G. Conforti, D. Macedonio, and V. Sassone. Spatial logics for bigraphs (extended abstract). In *Proc. of International Colloquioum on Automata and Languages (ICALP)*, 2005.

[62] S. Dal Zilio. Fixed points in the ambient logic, April 20 2001.

[63] S. Dal Zilio, D. Lugiez, and C. Meyssonnier. A logic you can count on. *ACM SIGPLAN Notices*, 39(1):135–146, January 2004.

[64] S. Dal Zilio and D. Luigez. Multitrees automata, presburger's constraints and tree logics. LIF Research Report 08-2002, 2002.

[65] S. Dal Zilio and D. Luigez. XML Schema, Tree Logic and Sheaves Automata. INRIA Research Report 4631, 2002.

[66] T.C. Damgaard and L. Birkedal. Axiomatizing binding bigraphs. Technical Report TR-2005-65, IT University of Copenhagen.

[67] Anuj Dawar and Giorgio Ghelli. Expressiveness and complexity of graph logic. Technical report, April 19 2004.

[68] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML, 1998. Submission to the W3C. http://www.w3.org/TR/NOTE-xml-ql.

[69] W. Fan and J. Siméon. Integrity constraints for XML. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS-00)*, pages 23–34. ACM Press, May 15–17 2000.

[70] M. Gabbay and A.M. Pitts. A new approach to abstract syntax involving binders. In *Proc. of LICS'99*, pages 214–224. IEEE Computer Society Press, 1999.

[71] P. Gardner and S. Maffeis. Modelling dynamic web data. *Theoretical Computer Science, to appear*, 2004.

[72] Philippa Gardner and Lucian Wischik. Explicit fusions. In Mogens Nielsen and Branislav Rovan, editors, *Mathematical Foundations of Computer Science , 25th International Symposium, MFCS 2000 (Bratislava, Slovakia)*, volume 1893 of *LNCS*, pages 373–382. Springer, 2000.

[73] T. Hildebrandt, H. Niss, M. Olsen, and J. W. Winther. Distributed reactive xml, an xml-centric coordination middleware.

[74] D. Hirschkoff. An extensional spatial logic for mobile processes. In *Proc. of CONCUR*, pages 325–339, 2004.

[75] H. Hosoya and B. Pierce. Regular expression pattern matching for XML. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 2001)*, pages 67–80. ACM Press, 2001.

[76] H. Hosoya and B.C. Pierce. XDuce: A typed XML processing language (preliminary report). In *Proc. of Workshop on the Web and Data Bases (WebDB)*, volume 1997 of *LNCS*, pages 226–244. Springer-Verlag, 2001.

[77] H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for XML. In *Proceedings of the ACM Sigplan International Conference on Functional Programming (ICFP-00)*, volume 35.9 of *ACM Sigplan Notices*, pages 11–22, N.Y., September 18–21 2000. ACM Press.

[78] S. Isthiaq and P.W. O'Hearn. BI as assertion language for mutable data structures. In *Conference Record of 28th ACM POPL*, 2002.

[79] O. H. Jensen and R. Milner. Bigraphs and transitions. In *Proc. of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 38–49. ACM Press, 2003.

[80] O. H. Jensen and R. Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580. University of Cambridge, February 2004.

[81] O.H. Jensen. Forthcoming PhD Thesis. Aalborg University, 2004.

[82] N. Klarlund, A. Möller, and M. I. Schwartzbach. DSD: A schema language for XML. In Mats P. E. Heimdahl, editor, *Proceedings of the 3rd Workshop on Formal Methods in Software Practice (FMSP-00)*, pages 101–111, N. Y., August 24–25 2000. ACM Press.

[83] D. Kozen. Results on the propositional mu -calculus. *Theoretical Computer Science*, 27(3):333–354, December 1983.

[84] A. Kwong and M. Gertz. Schema based optimization of XPath expressions. Submitted for publication, available from the authors, 2002.

[85] A. Kwong and M. Gertz. Structural constraints for XML. Technical Report CSE-2002-24, extended abstract submitted for publication, 2002.

[86] Yves Lafont. Interaction nets. In ACM, editor, *POPL '90. Proceedings of the seventeenth annual ACM symposium on Principles of programming languages, January 17–19, 1990, San Francisco, CA*, pages 95–108, New York, NY, USA, 1990. ACM Press.

[87] D. Lee and W. W. Chu. Comparative analysis of six XML schema languages. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(3):76–87, September 2000.

[88] Leifer and Milner. Deriving bisimulation congruences for reactive systems. In *CONCUR: 11th International Conference on Concurrency Theory*. LNCS, Springer-Verlag, 2000.

[89] E. Lozes. Expressivité des logiques d'espaces. Ph.D. Thesis. Ecole Normale Supérieure de Lyon, September 2004.

[90] E. Lozes. Elimination of spatial connectives in static spatial logics. *TCS: Theoretical Computer Science*, 330, 2005.

[91] Saunders Mac Lane. *Categories for the working mathematician*. Graduate Texts in Mathematics. Springer, New York / Berlin, 2nd. edition edition, 1998.

[92] R. Milner. Calculi of interaction. *Acta Informatica*, 33:707–737, 1996.

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

[94] R. Milner. Axioms for bigraphical structure. Technical Report UCAM-CL-TR-581. University of Cambridge, February 2004.

[95] R. Milner. Bigraphs for petri-nets. In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, pages 686–701. Springer, 2004.

[96] R. Milner. Pure bigraphs. Technical Report UCAM-CL-TR-614. University of Cambridge, January 2005.

[97] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, pages 1–40 & 41–77, September 1992. Tech' rep's LFCS-89-85 and LFCS-89-86.

[98] A. Muscholl, T. Schwentick, H.Seidl, and P. Habermehl. Counting in trees for free. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 2004.

[99] Shane O'Conchuir. Kind bigraphs-static theory. Technical Report, Trinity College, Dublin, 2005.

[100] Peter O'Hearn. Resources, concurrency and local reasoning. To appear in Theoretical Computer Science, preliminary version in CONCUR'04, 2005.

[101] Peter O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *In Proc. of CSL*, 2001.

[102] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proceedings of the Nineteenth ACM Symposium on Principles of Database Systems (PODS-00)*, pages 35–46. ACM Press, 2000.

[103] A. M. Pitts. Nominal logic: A first order theory of names and binding. In *Proc. of TACS 2001*, volume 2215 of *LNCS*, pages 219–242. Springer-Verlag, 2001.

[104] D.J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*. Kluwer Academic Publishers, 2002.

[105] John C. Reynolds. Intuitionistic reasoning about shared mutable data structures. *Millenial Perspectives in Computer Science*, pages 303–321, 2000.

[106] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of 17th IEEE Symponsium on Logic in Computer Science*, 2002.

[107] D. Sangiorgi. Extensionality and intensionality of the ambient logics. In *Proc. of POPL*, volume 36 of *ACM SIGPLAN Notices*, pages 4–13. ACM Press, 2001.

[108] H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In ACM, editor, *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems: PODS 2003: San Diego, Calif., June 9–11, 2003*, pages 155–166, New York, NY 10036, USA, 2003. ACM Press.

[109] Helmut Seidl, Thomas Schwentick, and Anca Muscholl. Numerical document queries. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 155–166. ACM, 2003.

[110] Helmut Seidl, Thomas Schwentick, Anca Muscholl, and Peter Habermehl. Counting in trees for free. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 1136–1149. Springer, 2004.

[111] V. Vianu. A Web odyssey: from Codd to XML. In *Proc. of the 20th ACM Symposium on Principles of Database Systems: PODS 2001*, SIGMOD Record, pages 1–15. ACM Press, 2001.

[112] Bjorn Victor and Joachim Parrow. The fusion calculus: Expressiveness and symmetry in mobile processes, December 05 1997.

[113] Pawel Wojciechowski and Peter Sewell. Nomadic pict: Language and infrastructure design for mobile agents. In *First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)*, Palm Springs, CA, USA, October 1999.

[114] H. Yang. An example of local reasoning in BI pointer logic: The Schorr-Waite graph marking algorithm. In *Informal Proc. of SPACE*, 2001.

[115] Hongseok Yang and Peter O'Hearn. A semantic basis for local reasoning. In *Proc. of FOSSACS*, 2002.