Models of Sharing Graphs

A Categorical Semantics of let and letrec

Masahito Hasegawa

Doctor of Philosophy
University of Edinburgh
1997

To my parents

Abstract

A general abstract theory for computation involving shared resources is presented. We develop the models of *sharing graphs*, also known as term graphs, in terms of both syntax and semantics.

According to the complexity of the permitted form of sharing, we consider four situations of sharing graphs. The simplest is first-order acyclic sharing graphs represented by let-syntax, and others are extensions with higher-order constructs (lambda calculi) and/or cyclic sharing (recursive letrec binding). For each of four settings, we provide the equational theory for representing the sharing graphs, and identify the class of categorical models which are shown to be sound and complete for the theory. The emphasis is put on the algebraic nature of sharing graphs, which leads us to the semantic account of them.

We describe the models in terms of the notions of symmetric monoidal categories and functors, additionally with symmetric monoidal adjunctions and traced monoidal categories for interpreting higher-order and cyclic features. The models studied here are closely related to structures known as notions of computation, as well as models for intuitionistic linear type theory. As an interesting implication of the latter observation, for the acyclic settings, we show that our calculi conservatively embed into linear type theory. The models for higher-order cyclic sharing are of particular interest as they support a generalized form of recursive computation, and we look at this case in detail, together with the connection with cyclic lambda calculi.

We demonstrate that our framework can accommodate Milner's *action calculi*, a proposed framework for general interactive computation, by showing that our calculi, enriched with suitable constructs for interpreting parameterized constants called controls, are equivalent to the closed fragments of action calculi and their higher-order/reflexive extensions. The dynamics, the computational counterpart of action calculi, is then understood as rewriting systems on our calculi, and interpreted as local preorders on our models.

Acknowledgements

I want to express my heartfelt thanks to my supervisors; I can never thank them enough.

Rod Burstall, my first supervisor, always helped me to think constructively and positively, especially at difficult moments throughout my PhD study in Edinburgh. I will never forget a meeting with Rod when I had a bad hangover – there I got an essential inspiration in deciding my research direction.

Philippa Gardner, my second supervisor during the second year, always gave me enthusiastic encouragement, and I benefited from countless delightful (often overheated) discussions with her.

John Power has always been an important intellectual source and often a mentor for me during these three years, and he became my second supervisor after Philippa moved to Cambridge. Without his generous and much needed support, this thesis would probably never have been written in this form.

At a stimulating place like LFCS, even a brief chat often meant a lot to me. I am grateful to people who influenced me in various forms, especially to Andrew Barber, Ewen Denney, Marcelo Fiore, Alex Mifsud, Robin Milner, Gordon Plotkin and Alex Simpson. In particular, Chapter 5 and Chapter 8 refer to joint work with Andrew, Philippa and Gordon.

I want to thank Martin Hyland for helpful discussions on traced monoidal categories as well as for his warm encouragement. Thanks are also due to Zena Ariola and Stefan Blom, for e-mail communications on cyclic lambda calculi and related topics.

And, above all, many, many thanks go to my cheerful, lovely, friends.

This work was partly supported by an Oversea Research Student award.

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Masahito Hasegawa*)

Table of Contents

Chapter 1 Introduction **3**

1.1 Computation Involving Shared Resources 3

1.2 Sharing Graphs as Models of Sharing 4

1.3 Sharing Graphs and Their Presentation 7

1.4 Categorical Models for Sharing Graphs 10

1.5 Relating Models 14

1.6 Recursion from Cyclic Sharing 15

1.7 Action Calculi as Graph Rewriting 16

1.8 Overview 18

Chapter 2 Sharing Graphs and Equational Presentation **20**

2.1 Sharing Graphs 20

2.2 Acyclic Sharing Theory 25

2.3 Cyclic Sharing Theory 34

2.4 Rewriting on Sharing Graphs 38

2.5 Equational Term Graph Rewriting 40

Chapter 3 Models of Acyclic Sharing Theory **42**

3.1 Preliminaries from Category Theory 43

3.2 Acyclic Sharing Models 47

3.3 The Classifying Category 54

3.4 Theory-Model Correspondence 59

3.5 Modeling Rewriting via Local Preorders 60

Chapter 4 Higher-Order Extension **62**

4.1 Higher-Order Acyclic Sharing Theory 63

4.2 Higher-Order Acyclic Sharing Models 64

4.3 The Classifying Category 68

Chapter 5 Relating Models **71**

5.1 Preliminaries from Category Theory 71
5.2 Higher-Order Extension 73
5.3 Notions of Computation 74
5.4 Models of Intuitionistic Linear Logic 76

Chapter 6 Models of Cyclic Sharing Theory **77**

6.1 Traced Monoidal Categories 77
6.2 Cyclic Sharing Models 82
6.3 The Classifying Category 86

Chapter 7 Recursion from Cyclic Sharing **89**

7.1 Fixed Points in Traced Cartesian Categories 90
7.2 Generalized Fixed Points 93
7.3 Higher-Order Cyclic Sharing Theory 98
7.4 Cyclic Lambda Calculi 102
7.5 Analyzing Fixed Points 108

Chapter 8 Action Calculi **110**

8.1 Action Calculi: Definitions, Basics 110
8.2 Action Calculi as Sharing Theories 113
8.3 Extensions 116

Chapter 9 Conclusion **123**

Appendix A Proofs **126**

A.1 Proof of Proposition 6.1.5 126
A.2 Proof of Theorem 7.1.1 127
A.3 Proof of Theorem 7.2.1 131
A.4 Proof of Proposition 7.1.4 132
A.5 Proof of Proposition 7.2.2 134

Bibliography **135**

Chapter 1

Introduction

1.1 Computation Involving Shared Resources

The notion of *sharing* has appeared on various occasions in computer science, either explicitly or implicitly. The idea is simple: instead of giving computational resources (processes, memories etc) to each client, a single resource can be shared by multiple clients.

In general, this kind of replacement may change the nature of the involved computation significantly. For instance, if the resource we are concerned with requires heavy computation or a large memory, sharing becomes an essential technique for saving both time and space needed for the computation. Many implementations of pure functional programming languages are based on this observation – avoiding unnecessary duplication of subcomputation is crucial for achieving efficient functional computation.

However, sharing is not just about the efficiency. If the resource involves some computation with side effects, say non-determinism or imperative states, the sharing of such a resource may change not just the amount of computation but also the result of computation. In such impure cases, the distinction between duplicated resources and shared resources must be made more carefully, and this makes it difficult, or at least non-trivial, to reason about general computation involving shared resources.

Furthermore, sharing can naturally be used for implementing *cyclic* (self-referential) data structures, which have been used for implementing recursive computation efficiently. The expressive power obtained by cyclic sharing is enormous, but dealing with cyclic structures is far more difficult than dealing with just acyclic ones. For instance, there are various practical ways of encoding recursive computation using cyclic sharing, but, to the best of our knowledge, there has been no formal comparison between them.

This thesis is devoted to giving a theory for describing and reasoning about such computation with sharing. The weight is put on the study of the *classes* of models of sharing, rather than individual specific models, in a desire to extract a generic account for sharing.

1.2 Sharing Graphs as Models of Sharing

Sharing for Efficiency

No programmer would be happy to write an expression like

```
... (factorial(100) + 123) * factorial(100) ...
```

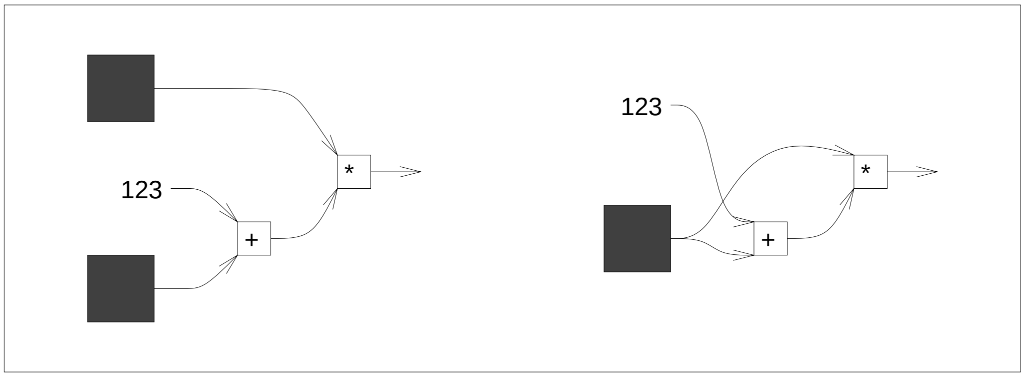
containing two identical subexpressions `factorial(100)`, not just because it makes the program messy but because it does suggest a duplication of very heavy computation (here we suppose that the program `factorial(100)` calculates the factorial of 100, which in many cases results in an overflow). The former reason may be very important from the view of software engineering where readability and reusability of programs are essential, but it is not a matter to be discussed now. Here we shall stick to the second point - efficiency. Many programmers should agree to rewrite the expression above as

```
let x = factorial(100) in ... (x + 123) * x ...
```

The intention is that, we avoid calculating `factorial(100)` twice by *sharing* the result of this computation, without changing the result of computation. The `let` syntax indicates that `factorial(100)` is a shared resource with a name `x` which are later referred (used) at two places in the program.

But actually this is not just a matter for programmers, but more essentially the problem of the implementor of the programming language. Though the two examples above are supposed to return the same result, hence are extensionally equivalent, they are "intensionally" different because the amount of the involved computation is different; implementors must realize some semantic models in which such these two have distinct denotations – they may not be models for programmers (who just care about the results) but are models for implementors (who care about the actual computational steps behind the results).

Graph rewriting theory – the theory of *sharing graphs* (*term graphs*) and rewriting systems on them – has been recognized as a canonical and practically useful instance of such models for implementors [BvEG+87, SPvE93]. The idea is to use graphs for representing the sharing relations of resources and realize computation on them as rewriting systems. For instance, the first example can be explained simply by the graphical representation of the expressions, as



The left tree corresponds to the original unshared version, whereas the right graph is for the "refined" version with sharing of a resource. The actual computation is modeled by rewriting, i.e. local replacement of subgraphs. Obviously the left one requires more computation (rewriting steps) because of the duplicated resource (subgraph).

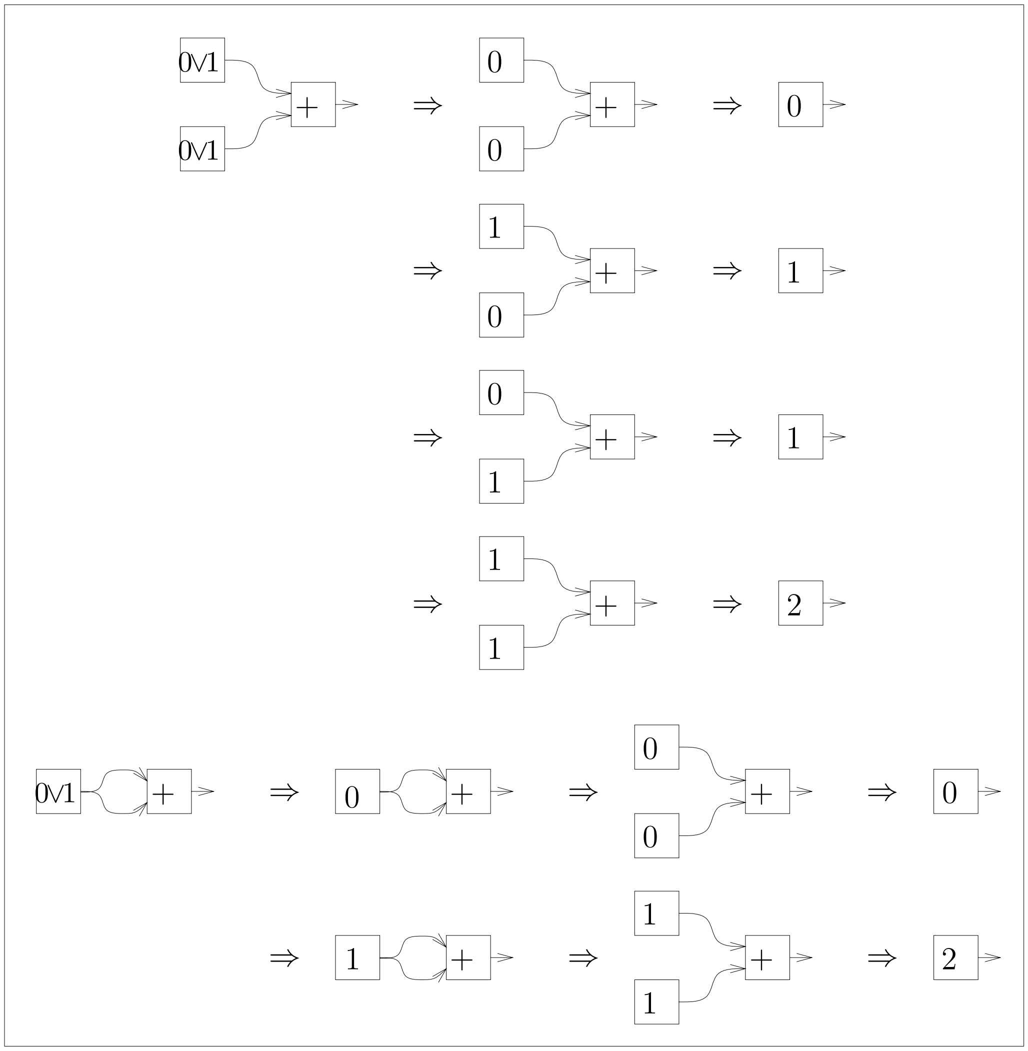
Impure Cases: Sharing as a Programming Technique

Consider a language with a non-deterministic construct `zeroORone` which returns 0 or 1 at random. As before, we shall use the let-syntax for representing sharing. Then the following two programs obviously have the different meanings.

```
zeroORone + zeroORone

let x = zeroORone in x + x
```

The former returns 0, 1 or 2, whereas the latter 0 or 2 (see the picture below). In this case the shared resource is not *pure*; it contains a side-effect, thus should be better understood as a process in a concurrent language or an object in an object-oriented language. Similar things happen if we consider imperative languages with states. In such "impure" settings, introducing sharing may change the result of computation, hence changing the extensional (programmers') semantics of the language. Therefore sharing becomes an important feature of the programming language which programmers have to recognize as a programming technique; and actually most programmers of impure languages do, often explicitly when manipulating states, objects and memories.



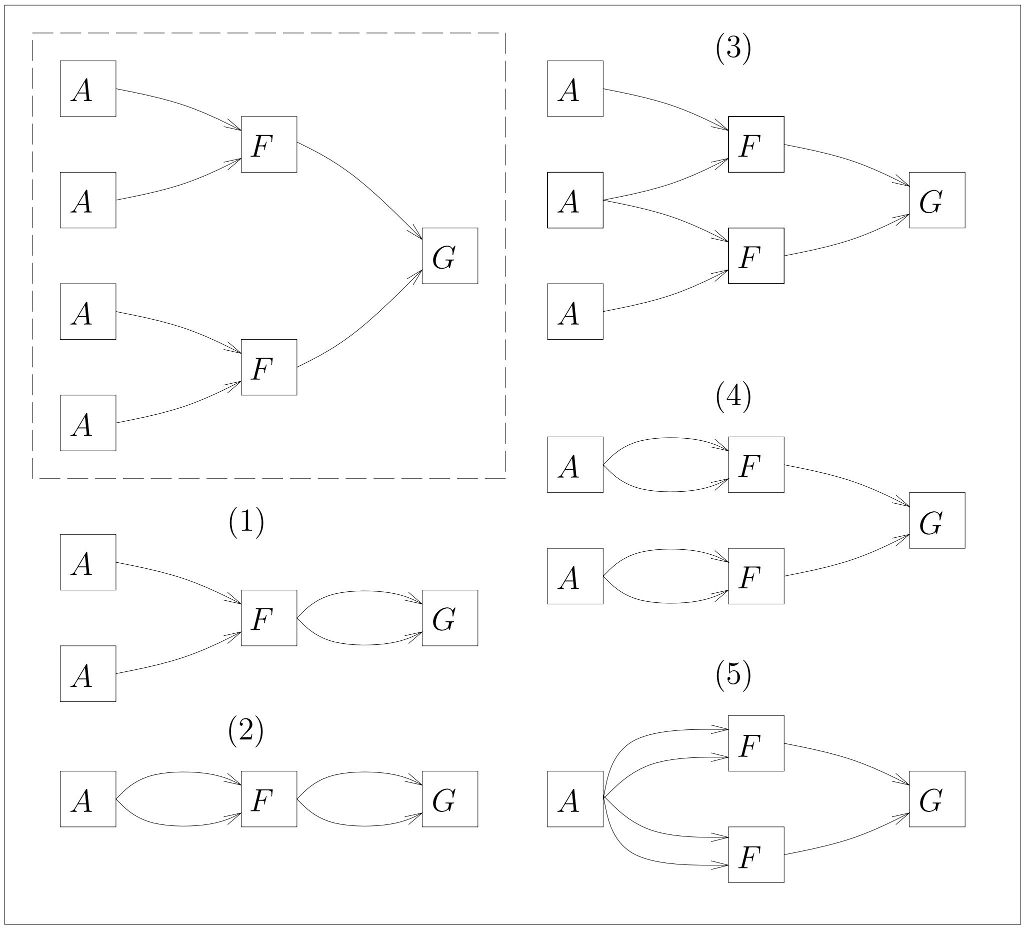
Cyclic Sharing and Recursion

Circular phenomena have been a rich source of a wide range of intellectual investigations for long time – in science, technology, and even philosophy; see [BM96] for a survey and lots of examples. Computer science is not an exception. Sharing graph-based models have a natural advantage in representing cyclic data structures, and the most interesting and practical usage of such cyclic sharing is, of course, as the means of realizing recursive computation, which is one of the most important subjects in computer science. As already shown by Turner [Tur79] in 70s, recursive computation can be efficiently implemented using self-referential (i.e. cyclic) terms. We come back this point later and explain in some detail – the analysis of recursive computation created from cyclic sharing is one of the central implications of this thesis.

1.3 Sharing Graphs and Their Presentation

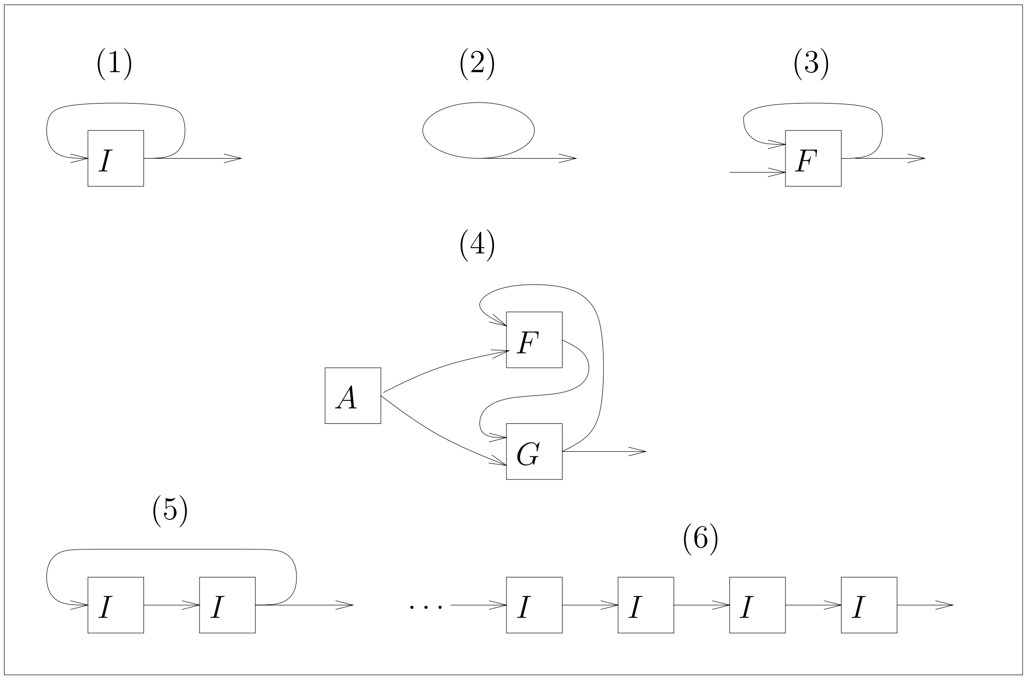
As motivated above, we regard sharing graphs, or term graphs, as abstract representations of the sharing relation of resources. They can be seen as a special sort of directed graphs in which nodes represent resources and links show the sharing, but perhaps better understood as a generalization of the tree notations for terms - the name "term graphs" means the direct generalization of "term trees".

If there is no notion of sharing, it suffices to talk about just trees (terms) where subtrees (subterms) correspond to subcomputations. However, if we want to talk about sharing, trees are not sufficient, and we are naturally lead to replace trees by a class of directed graphs. Now a subgraph may be referred from various places in the graph, thus representing a shared resource. The pictures below show that there are various sharing graphs corresponding to a term $G(F(A, A), F(A, A))$.



As mentioned earlier, the meaning of sharing changes depending on the computation concerned. If each node represents purely functional computation, the difference between these sharing graphs is just about the amount of computation. The final answer will be the same, but the sharing graph (2) presents the optimal way to get the answer. On the other hand, if A is a process which returns 0 or 1 non-deterministically and F and G calculate the sum of arguments, then the original term presents a computation which returns 0, 1, 2, 3 or 4, while (1) and (4) return 0, 2 or 4, whereas (2) and (5) returns just 0 or 2. (3) returns 0, 1, 2, 3 or 4 as the original term, but the probability would be changed.

Allowing cyclic bindings, sharing graphs get further flexibility. Let us look at some instances of cyclic sharing graphs.



(1) and (2) present the simplest situations of cyclic sharing. In (1), the resource I refers to itself; (2) may seem odd as it does not involve any resource, but such a "self-referential pointer" or "trivial cycle" can occur even in a realistic situation. (3) is similar to (1), except that it has one additional input. A more sophisticated example is (4) where F and G mutually refer each other. (1) and (5) have the same tree-unwinding $I(I(I(I(\ldots))))$ as (6), but again it depends on the situation whether we should identify the meaning of (1) and (5) with (6).

Now we turn our attention to how to present term graphs concisely. Defining them as directed graphs, as we will do later, is not very informative; sharing graphs have more structural and algebraic properties than general directed graphs do, and we wish to capture this nature. A first hint comes from the observation above that sharing graphs can be obtained by enriching traditional terms (trees) with constructs for acyclic or cyclic sharing. Our programming example already suggests a convenient syntax for them - the let (letrec) blocks.

Actually similar notions have appeared in many places for presenting similar kind of (possibly circular) dependency relations. There are various versions of *systems of equations* for describing "non-well-founded sets" [Acz88, BM96] like

$$\begin{array}{rcl} x &=& \{y\} \\ y &=& \{x, z\} \\ z &=& \{x\} \end{array}$$

(The anti-foundation axiom states that this kind of system has a unique solution.) Similarly it is common to present a state transition system like finite state automata, and also concurrent processes, e.g. [Mil89], by a system of equation

$$\mathit{Clock} \quad = \quad \mathit{tick}.\mathit{Clock} + \mathit{break}.\mathit{Stuckclock}$$

Yet another popular instance is the description of inductive (or recursive) types: for instance the type T of finite branching finite trees can be represented as a solution of a system of equations

$$\begin{array}{rcl} T &=& F \\ F &=& 1 \,+\, T \times F \end{array}$$

(The terms can be generated by BNF

$$\begin{array}{rcl} t &::=& \mathtt{span}(f) \\ f &::=& \mathtt{nil} \mid \mathtt{cons}(t, f)\ .) \end{array}$$

These systems of equations have natural graph presentations, though it is possible that two different systems may describe the identical graph[1]. So there should be an equational theory on these systems which is sound and complete with respect to the graph interpretation.

We give such an axiomatization on our terms with the let/letrec blocks (which are of course an instance of systems of equations). Such notation has an advantage in allowing us equational and inductive structural reasoning about sharing graphs. We inductively construct (the presentations of) sharing graphs from variables (pointer names), function symbols (resources) and systems of equations. Thus, as the traditional algebraic theories for terms, we give equational theories for sharing graphs in terms of systems of equations for which we use the let/letrec-binding syntax. For instance, the acyclic sharing graphs in the first example can be presented as

$$\begin{array}{ll} (1) & \mathsf{let}\ x = F(A, A)\ \mathsf{in}\ G(x, x) \\ (2) & \mathsf{let}\ y = A\ \mathsf{in}\ \mathsf{let}\ x = F(y, y)\ \mathsf{in}\ G(x, x) \\ (3) & \mathsf{let}\ y = A\ \mathsf{in}\ G(F(A, y), F(y, A)) \\ (4) & \mathsf{let}\ y = A\ \mathsf{in}\ \mathsf{let}\ y' = A\ \mathsf{in}\ \mathsf{in}\ G(F(y, y), F(y', y')) \\ (5) & \mathsf{let}\ y = A\ \mathsf{in}\ G(F(y, y), F(y, y)) \end{array}$$

[1]Actually, for these examples, we usually work up to some stronger equivalences than that of graphs; for instance two systems are often equated if they correspond to the same infinite unwinding, equivalently if they are "bisimular". But here we do not presuppose such specific semantic interpretations, and just compare the graphs concerned themselves.

As noted above, two different terms can represent the same graph; for instance, (3) can be presented as $\mathsf{let}\ y = A\ \mathsf{in}\ \mathsf{let}\ x = F(A, y)\ \mathsf{in}\ G(x, F(y, A))$, and our equational theory guarantees that this is equal to $\mathsf{let}\ y = A\ \mathsf{in}\ G(F(A, y), F(y, A))$. Similarly, the (finite) cyclic sharing graphs in the second picture correspond to

(1) $\mathsf{letrec}\ x = I(x)\ \mathsf{in}\ x$
(2) $\mathsf{letrec}\ x = x\ \mathsf{in}\ x$
(3) $\mathsf{letrec}\ x = F(y, x)\ \mathsf{in}\ x$
(the free variable y represents the unspecified input node)
(4) $\mathsf{letrec}\ x = A,\ y = F(x, z),\ z = G(x, y)\ \mathsf{in}\ z$
(5) $\mathsf{letrec}\ x = I(I(x))\ \mathsf{in}\ x$

A simple discipline of typing is naturally given, as for traditional algebraic theories, in which any sharing graph is equipped with its input and output types (sorts). This allows us to construct graphs by well-typed composition inductively.

Moreover, the rewriting rules on sharing graphs are easily presented on such an equational formulation, in similar manner to the usual term rewriting rules on algebraic theories. The only difference is that in each rewriting step we replace a subgraph by another (with the same typing), instead of replacing a subterm by another.

Such advantages of this style of presentation have already been emphasized and studied by Klop, Ariola and others in the context of graph rewriting theory [AA95, AK96]. In this thesis we basically follow their ideas, but use them freely in a more general and semantic (algebraic) context. The merit of the equational presentation becomes clearer in developing the semantic counterpart of sharing graphs, as explained below.

1.4 Categorical Models for Sharing Graphs

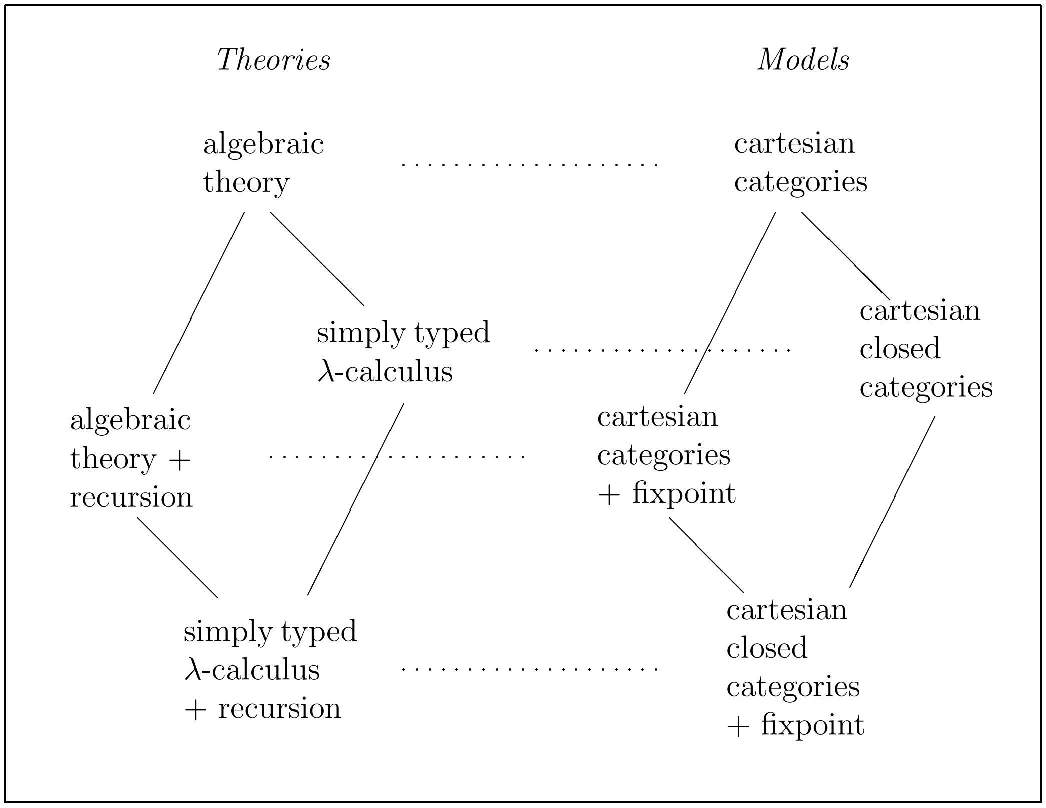
Traditionally, the semantic account of sharing graphs has been given in specific models, most importantly as tree unwindings where two sharing graphs are identified if they represent the same (possibly infinite) tree. Such a semantics stands out if we use sharing graphs for representing efficient implementations of pure functional computation. In this thesis, however, we take a different starting point, for the following reasons.

1. We wish to keep as many choices of semantic models as possible, so that we can interpret various (impure) forms of computation flexibly. For instance, if we want to take non-determinism into account, the infinite tree unwinding semantics is inconsistent. Rather than starting from specific models and

trying to interpret actual computation in them, we axiomatize the properties needed by the models of sharing, and then find intended models.

2. We wish to talk about the *class* of models. This enables us to prove general results on all models at once, and also to classify models in a natural manner. For instance, we will give relations between our sharing graphs and intuitionistic linear logic by comparing the classes of models.

For describing the classes of models of sharing graphs, we find category-theoretical languages useful. The canonical examples of the use of category theory in this direction are the correspondence between algebraic theories and cartesian categories (categories with finite products), as well as that between the simply typed lambda calculus and cartesian closed categories. Let us summarize these "standard" *categorical type theory* correspondence as below; to make the connection with cyclic sharing, we include the treatment of recursion in our picture.


Theories
Models
algebraic theory
cartesian categories
simply typed λ-calculus
cartesian closed categories
algebraic theory + recursion
cartesian categories + fixpoint
simply typed λ-calculus + recursion
cartesian closed categories + fixpoint

Following Lawvere [Law63], we give models of an algebraic theory by a *finite product preserving functor* from the *classifying category* (term model) of the algebraic theory into a cartesian category. Each function symbol F with arity $((\sigma_1, \ldots, \sigma_n), \tau)$ is interpreted as an morphism $[\![F]\!] : [\![\sigma_1]\!] \times \ldots \times [\![\sigma_n]\!] \to [\![\tau]\!]$ in the target cartesian category, where $[\![\sigma_i]\!]$, $[\![\tau]\!]$ present the objects associated with each sort σ_i, τ in the algebraic theory, and $\times$ is the (chosen) cartesian product. The interpretation is then inductively extended to all expressions (terms) in the algebraic theory – it determines a finite product preserving functor from the classifying category into the model category if and only if it satisfies the soundness property: if two expressions are provably equal in the theory, then their interpretation in the model is the same morphism. This is the basic picture of the theory-model correspondence in categorical type theory. A detailed account can be found, for instance, in [Cro93].

This basic setting can be enriched with higher-order features, as well as recursive computation. For shifting to the higher-order extension, we require the existence of exponents, thus assume that the functor $(-) \times X$ has a right adjoint for each object X in the model category. Therefore we are led to the notion of *cartesian closed categories*, and again we get the theory-model correspondence between simply typed lambda theories and cartesian closed categories [Cro93, LS86] (this time the semantic interpretations are given as cartesian closed functors).

For recursion, the standard way is to assume a construction on the model cartesian (closed) category, called a *(parameterized) fixed point operator*

$$\frac{f : A \times X \to X}{f^\dagger : A \to X}$$

which is subject to the condition that $\langle id_A, f^\dagger\rangle ; f = f^\dagger$ (to be more precise, we assume that this construction is natural in A, so that the model is sound for the interpretation of substitutions). In the standard notation for a recursion operator on algebraic theories, this corresponds to

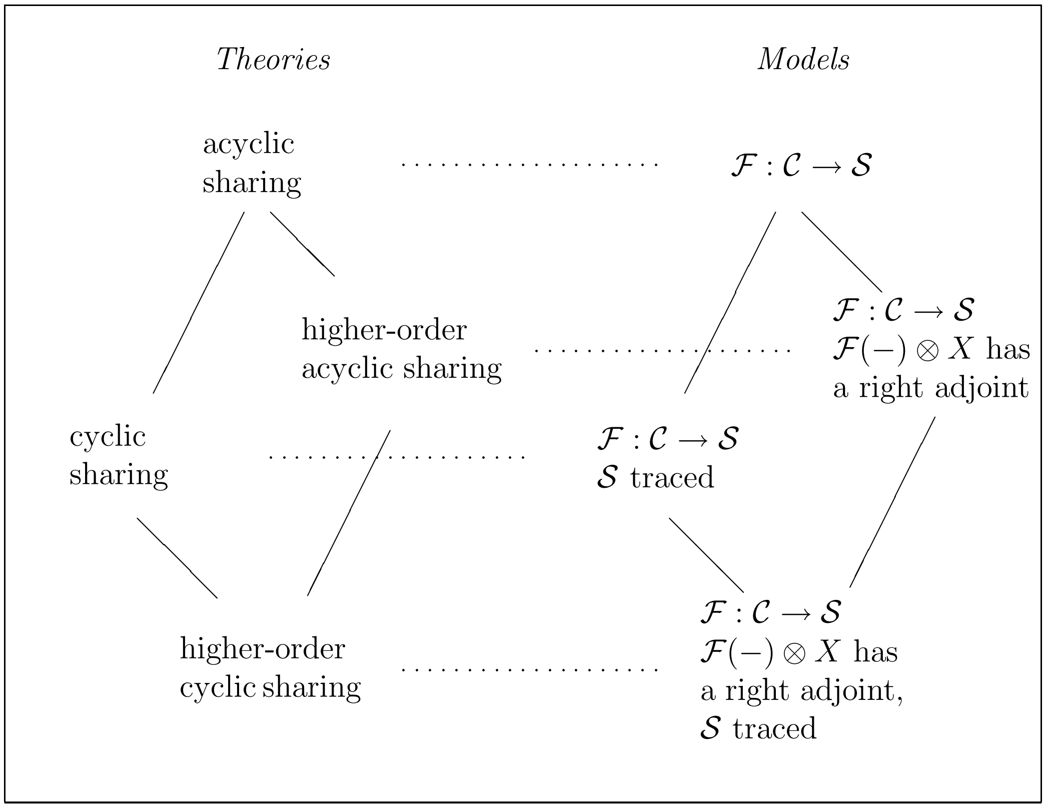
$$\frac{\Gamma, x : \sigma \vdash M : \sigma}{\Gamma \vdash \mu x.M : \sigma}$$

with the fix point equation $\mu x.M = M[\mu x.M/x]$. Many concrete examples of such categories are found in domain theory, where cartesian closedness and existence of fixed point operators are fundamental requirements for giving the denotational semantics of programming languages.

The main technical development in this thesis is to give, for sharing graphs, a precise analog of this standard categorical type theory. The equational theory presentation of sharing graphs via the let (letrec)-syntax is already very close to the standard algebraic theories, and it is natural to expect that there is a similar theory-model correspondence for sharing graphs.

The essential change is that, instead of cartesian categories, we take identity-on-objects, strict symmetric monoidal functors from cartesian categories to symmetric monoidal categories as the basic setting for interpreting the sharing graphs.

Intuitively, the domain cartesian category is used for modeling the non-linear nature of sharing graphs – pointer names, or links, and also copyable-values (if they exist), are duplicated or discarded freely, hence will be interpreted in the cartesian category as we do for algebraic theories. On the other hand, the codomain symmetric monoidal category is for interpreting linear entities in sharing graphs; since we do not duplicate or discard the shared resources which are expensive or contain some side effect, they must be treated linearly. (The reader familiar with linear logic [Gir87] may informally understand this by the analogy with the logical connectives & and $\otimes$ of linear logic; later we will give the precise connection between our models of sharing and those of propositional intuitionistic linear logic.) The strict functor between them is to relate these non-linear and linear natures. In short, the essence of models of sharing lies in the separation of non-linear and linear features which live at the same time in the notion of sharing. Now we shall give our picture of the theory-model correspondence for sharing graphs.



Theories
Models
acyclic sharing
$\mathcal{F} : \mathcal{C} \to \mathcal{S}$
higher-order acyclic sharing
$\mathcal{F} : \mathcal{C} \to \mathcal{S}$ $\mathcal{F}(-) \otimes X$ has a right adjoint
cyclic sharing
$\mathcal{F} : \mathcal{C} \to \mathcal{S}$ $\mathcal{S}$ traced
higher-order cyclic sharing
$\mathcal{F} : \mathcal{C} \to \mathcal{S}$ $\mathcal{F}(-) \otimes X$ has a right adjoint, $\mathcal{S}$ traced

By $\mathcal{F} : \mathcal{C} \to \mathcal{S}$, we mean an identity-on-objects strict symmetric monoidal functor $\mathcal{F}$ from a cartesian category $\mathcal{C}$ to a symmetric monoidal category $\mathcal{S}$.

For interpreting higher-order features, we additionally require that $\mathcal{F}(-) \otimes X$ has a right adjoint for each object X; this is the precise analog of cartesian closed categories for our setting. For interpreting cyclic sharing, we need a relatively new concept from category theory – *traced monoidal categories* [JSV96]. Intuitively, a traced symmetric monoidal category is a symmetric monoidal category equipped with a construct for "feedback", called a *trace*:

$$\frac{f : A \otimes X \to B \otimes X}{Tr^X_{A,B}(f) : A \to B}$$

It would be helpful to understand that, in $Tr^X(f)$, f's output X is feedbacked, or linked, to f's input X. The formal axiomatization for a trace will be recalled later; we will see that it precisely corresponds to the equivalence on cyclic graphs, and the theory-model correspondence will be extended to the cyclic settings comfortably by assuming that the symmetric monoidal category $\mathcal{S}$ is traced.

The rewriting theories on sharing graphs are then simply modeled by local-preorders on the symmetric monoidal category $\mathcal{S}$ of our models. Some graph rewriting systems, especially the *equational term graph rewriting* by Klop and Ariola, are close to our theories and their semantic models.

Note that if we restrict our attention to the case that $\mathcal{C}$ and $\mathcal{S}$ are the same cartesian category and $\mathcal{F}$ is the identity functor, then we recover the standard categorical type theory as sketched before (a connection between traces and fixed point operators will be established later).

1.5 Relating Models

To demonstrate the advantage of our generic approach, we shall relate some known systems and ours by comparing their classes of models. Many people have pointed out that term graphs have some similarity with Girard's *linear logic* [Gir87], in their resource-sensitive natures. Also it has been pointed that Moggi's computational lambda calculus [Mog88] looks like higher-order graph rewriting systems. We give some formal accounts to these intuitive understandings, by first relating the classes of models, and then relating the theories as a corollary.

A model of propositional intuitionistic linear logic may be described as a symmetric monoidal adjunction between a cartesian closed category and a symmetric monoidal closed category [Bar96, Ben95, Bie95]. It is easily seen that such a structure is essentially a special case of the structures we have for interpreting acyclic sharing graphs, as sketched above. Thus there is a sound translation from the equational theory of sharing graphs into that of intuitionistic linear type theory. But we can say more: this translation is conservative, thus a linear type theory is seen as a conservative extension of the theory of sharing graphs. To prove this, we use the standard model construction technique from category theory (Yoneda construction as the free symmetric monoidal cocompletion [IK86]).

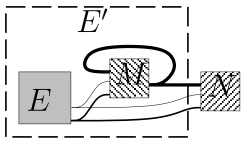
The connection with Moggi's work [Mog88, Mog91] is much more straightforward. The models for acyclic higher-order sharing will be shown to be essentially the same as his models for computational lambda calculus, with an assumption that the associated monad has a commutative strength. As a special instance of the theory developed by Power and Robinson [PR96, Pow96b], we describe this comparison.

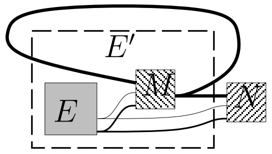
1.6 Recursion from Cyclic Sharing

One of the traditional methods of interpreting a recursive program in a semantic domain is to use the least fixed-point of continuous functions. However, as already mentioned, in the real implementations of programming languages, we often use some kind of shared cyclic structure for expressing recursive environments efficiently. For instance, the following is a call-by-name operational semantics of the recursive call, in which free x may appear in M and N. We write $E \vdash M \Downarrow V$ for saying that evaluating a program M under an environment E results a value V; in call-by-name strategy an environment assigns a free variable to a pair consisting of an environment and a program.

$$\frac{E' \vdash N \ \Downarrow\ V \quad \text{where } E' = E \cup \{x \mapsto (E', M)\}}{E \vdash \mathsf{letrec}\ x = M\ \mathsf{in}\ N \ \Downarrow\ V}$$

That is, evaluating a recursive program $\mathsf{letrec}\ x = M\ \mathsf{in}\ N$ under an environment E amounts to evaluating the subprogram N under a cyclic environment E' which references itself. One may see that it is reasonable and efficient to implement the recursive (self-referential) environment E' as a cyclic data structure as below.



or equivalently


Also it is known that if we implement a programming language using the technique of sharing, the use of the fixed point combinator causes some unexpected duplication of resources [AF96, Lau93]; it is more efficient to get recursion by cycles than by the fixed point combinator in such a setting. This fact suggests that there is a gap between the traditional approach based on fixed points and cyclically created recursion.

Our semantic models for higher-order cyclic sharing turn out to be the setting for studying recursive computation created by such a cyclic data structure, more specifically cyclic lambda graphs [AK94, AB97]. We claim that our new models are natural objects for the study of recursive computation because they unify several aspects on recursion in just one semantic framework. The leading examples are

- the *graphical (syntactical) interpretation* of recursive programs by cyclic data structures motivated as above,
- the *domain-theoretic interpretation* in which the meaning of a recursive program $\mathsf{letrec}\ x = F(x)\ \mathsf{in}\ x$ is given by the least fixed point $\bigcup_n F^n(\perp)$, and
- the *non-deterministic interpretation* where the program $\mathsf{letrec}\ x = F(x)\ \mathsf{in}\ x$ is interpreted by $\{x \mid x = F(x)\}$, the set of all possible solutions of the equation $x = F(x)$.

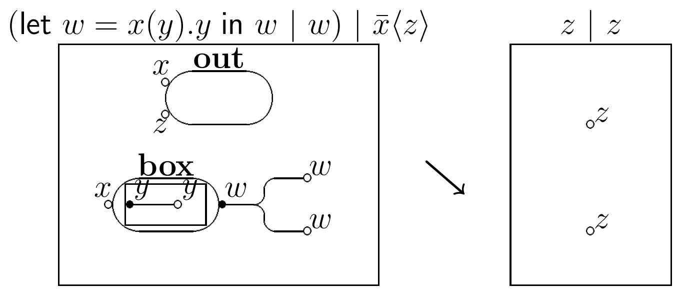
Each of them has its own strong tradition in computer science. However, to our knowledge, this is the first attempt to give a uniform account on these well-known, but less-related, interpretations. Moreover, our higher-order cyclic sharing theories and cyclic lambda calculi serve as a uniform language for them.

1.7 Action Calculi as Graph Rewriting

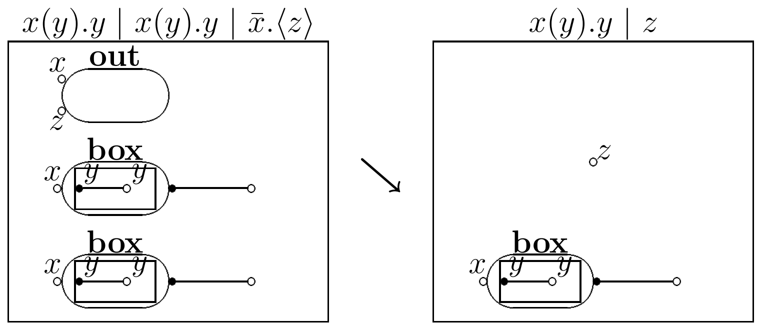
Finally we show that our framework can accommodate Milner's *action calculi* [Mil96], a proposed framework for general interactive computation, by showing that our sharing theories, enriched with suitable constructs for interpreting parameterized constants called controls, are equivalent to the closed fragments of action calculi [Gar95, Pow96a] and their higher-order/reflexive extensions [Mil94a, Mil94b, Mif96].

The dynamics, the computational counterpart of action calculi, is then understood as rewriting systems on sharing theories, and interpreted as local preorders on our models. In this sense, we understand action calculi as generalized graph rewriting systems – and regard the notion of sharing as one of the fundamental concepts of action calculi.

To demonstrate how sharing is used in action calculi, we shall consider two situations representable in the action calculus-version of the π-calculus [MPW92, Mil92a] as presented in [Mil96] (see Chapter 8, Example 8.1.6).


(let $w = x(y).y$ in $w \mid w) \mid \bar{x}\langle z\rangle$
$z \mid z$
out
box
x
z
y
w

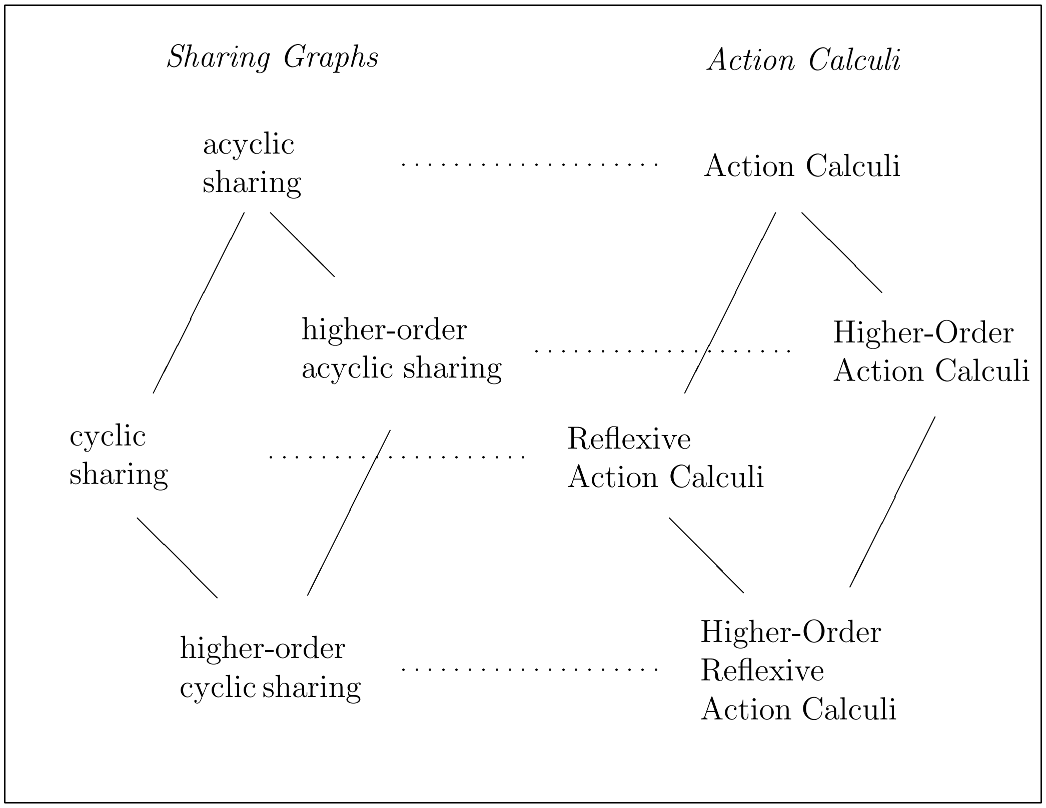
We may regard this situation (not representable in the original π-caclulus!) as a broadcasting; there is an announcer $x(y).y$ who gets a message via a telephone number x and then broadcasts it; her/his program is monitored by two listeners $w|w$. Therefore the received message z is broadcast (duplicated) to the listeners. Compare this and the unshared version $x(y).y \mid x(y).y \mid \bar{x}\langle z\rangle$, where we have two persons who share the same telephone number x. So we don't know which person will receive the message z, and there are two possible reactions (in both cases the result is $x(y).y \mid z$, thus one person remains unchanged:



Further sophisticated and complicated examples will be available by allowing cyclic sharing (reflexion) and higher-order constructions.

All of our semantic results on sharing graphs equally apply to action calculi (with some care on the treatment of controls). The conservativity of intuitionistic linear type theory over action calculi (as reported in [BGHP96, BGHP97]), the correspondence between higher-order action calculi and Moggi's work (as described in [GH97]), and the analysis of recursive computation in reflexive action calculi (c.f. [Mif96]) are obtained as corollaries of results on sharing graphs.

Here is a summary of the correspondence between our theory of sharing graphs and action calculi:


Sharing Graphs
Action Calculi
acyclic sharing
Action Calculi
higher-order acyclic sharing
Higher-Order Action Calculi
cyclic sharing
Reflexive Action Calculi
higher-order cyclic sharing
Higher-Order Reflexive Action Calculi

We hope that our work provide a bridge between graph rewriting theory and concurrency theory.

1.8 Overview

Chapter 2 introduces the notion of sharing graphs and the corresponding simply typed equational theories, called sharing theories. We emphasize the algebraic, structural nature of sharing graphs via the equational presentations, which leads us to the semantic development in the following chapters.

In Chapter 3 we study the category-theoretic models of acyclic sharing theories. In terms of symmetric monoidal categories and functors, we describe the class of models, and establish the soundness and completeness, in a similar way to the standard categorical type theory.

In Chapter 4 we give a higher-order extension of acyclic sharing. The models of this setting are obtained by assuming additional conditions formulated as adjunctions, and we repeat the same pattern as in Chapter 3.

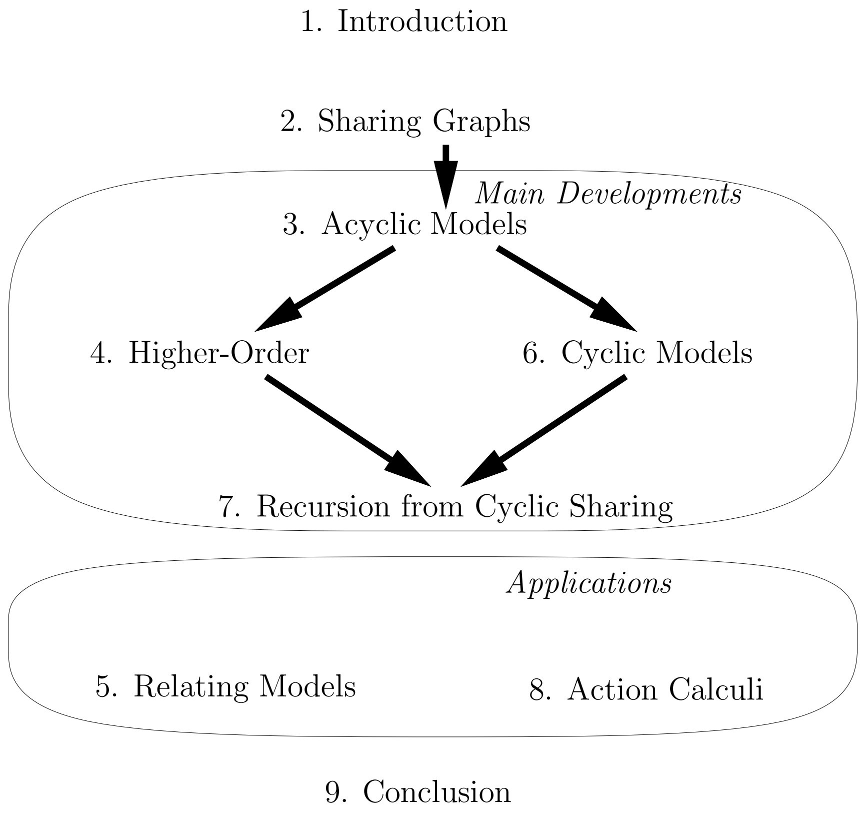
As an application of our approach, in Chapter 5 we relate our acyclic sharing theories with notions of computation and intuitionistic linear type theory by comparing their classes of models.

In Chapter 6 we give the models of cyclic sharing, by additionally using the notion of traced monoidal categories. After reviewing trace monoidal categories, we establish the expected properties of our models, again in the same way as Chapter 3.

Chapter 7 describes higher-order cyclic sharing. The models of this setting, obtained by combining those in Chapter 4 and Chapter 6, are of particular interest as they support a generalized form of recursive computation. We look at this in some detail, together with the connection with cyclic lambda calculi.

Chapter 8 is devoted to show that Milner's action calculi can be accommodated in our framework.

Finally, in Chapter 9, we conclude this thesis with some discussions towards further research.



Chapter 2

Sharing Graphs and Equational Presentation

Following the standard way of describing graphical structures, we first formulate sharing graphs as directed graphs with extra information and conditions, as found in the literature (though we give a mildly generalized version in which multiple roots, or conclusions, are permitted).

However, such a description is always lengthy, and technically not easy to manipulate. Also the algebraic, structural nature of sharing graphs is not clear in such formulations. Following an old idea of representing sharing graphs as systems of equations, we give an equational, in other words algebraic, presentation of finite sharing graphs.

Just in the same way that trees are represented as term expressions in algebraic theories, our sharing graphs are represented as term expressions of mildly relaxed algebraic theories enriched with constructs for sharing (let/letrec bindings). We establish the desired equivalence between the graph-theoretical description and the equational presentation of finite sharing graphs, for both acyclic and cyclic cases.

The results in this chapter are conceptually and technically not new at all – similar ideas have been around for long time. However, our intention here is to use the equational presentation of sharing graphs for emphasizing the algebraic aspects of them, which later naturally lead us to the formulation of the semantic models of sharing graphs. Therefore this chapter should be read as a preparation for our main technical developments.

2.1 Sharing Graphs

We first fix a signature on which our sharing graphs are constructed:

Definition 2.1.1 (signature)
Let S be a set of sorts. An (finitary) *S-sorted signature* is a set Σ of operation symbols together with an arity function assigning to each operation symbol F a pair of finite lists $((\sigma_1, \ldots, \sigma_m), (\tau_1, \ldots, \tau_n))$ of S's elements. Notation:

$$F : (\sigma_1, \ldots, \sigma_m) \longrightarrow (\tau_1, \ldots, \tau_n).$$

$\Box$

Remark 2.1.2 The main difference between our definition of signatures and more traditional one is that we allow multiple conclusions of an operator symbol. Intuitively, we allow operators which return tuples of results; or even operators which do not return anything (which, however, does not mean that such operators do not have computational significance – in some impure settings, they may have some side effects!). If the reader is familiar with algebraic theories with cartesian product types, this formulation could be considered as a mild variant, though our multiple conclusions will not form cartesian products, but symmetric monoidal products. So, while in the standard algebraic theories a term of product types can be regarded as a tuple of terms, this is not the case for our sharing graphs, where an "indecomposable" resource can have multiple outputs. $\Box$

For $F : (\sigma_1, \ldots, \sigma_m) \longrightarrow (\tau_1, \ldots, \tau_n)$, we may write $\mathrm{dom}(F) = (\sigma_1, \ldots, \sigma_m)$, $\mathrm{cod}(F) = (\tau_1, \ldots, \tau_n)$, $\mathrm{dom}(F)_i$ for σ_i (ith input sort) and $\mathrm{cod}(F)_j$ for τ_j (jth output sort).

In formulating (cyclic) sharing graphs, we need care about *trivial cycles* (also known as "blackholes") as repeatedly pointed in the literature of graph rewriting theory (in connection with the "cyclic-I problem", which will be discussed in Example 2.4.3), see e.g. [AK96]. A trivial cycle is a pointer which does not refer any resource but *itself* – just like a program "letrec x be x in x" which does not involve any computational resource but represents a circularly bound pointer. For both practical and technical reasons, we do not want to exclude such trivial cycles from our sharing graphs. To accommodate them, we need an additional constant for each sort:

Definition 2.1.3 (signature with $\bullet$)
Given an S-sorted signature Σ, we define an S-sorted signature $\Sigma_\bullet$ by additionally assuming an operation symbol $\bullet^\sigma : () \rightarrow (\sigma)$ for each sort σ. $\Box$

A rooted directed graph (with a label on each node) is specified by a set of nodes V, a labeling function L from V to the set of labels and a set of edges

$E \subseteq V \times V$ together with a specified root node $c \in V$. For describing our sharing graphs, we need more information as follows. First, an operation symbol may have multiple inputs and outputs, so we need to specify which input is linked to which output, in terms of an "argument function" A. Second, since we allow multiple outputs of the graph, we have to specify a list of outputs $(c_1, \ldots, c_n)$, rather than a single root node c. Thus we have a tuple $(V, L, A, (c_1, \ldots, c_n))$. Moreover, we want to make the graph well-typed, that is, an input and an output can be linked only when they have the same sort. So we assume a constraint for ensuring the well-typedness. Formally:

Definition 2.1.4 (sharing graph)
We fix a countable set $\{d_1, d_2, \ldots\}$. A (finitary) *sharing graph* over an S-sorted signature Σ of type $(\sigma_1, \ldots, \sigma_m) \to (\tau_1, \ldots, \tau_n)$ is a tuple $(V, L, A, (c_1, \ldots, c_n))$ such that

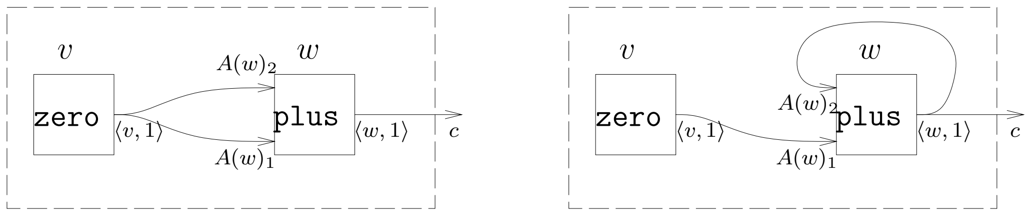
- V is a set.
- L is a function from V to $\Sigma_\bullet$.
- A is a function from V to C^*, such that $|A(v)| = |\mathrm{dom}(L(v))|$, where $C = \{\langle v, j\rangle \mid v \in V, 1 \leq j \leq |\mathrm{cod}(v)|\} \oplus \{d_1, \ldots, d_m\}$. Write $A(v)_i$ for the ith component of $A(v)$.[1] C serves as the set of all outputs (codomains), while d_i is just a name for the i-th input (domain).[2]
- $c_i \in C$ for $1 \leq i \leq n$.
- Condition on types:
  - For $1 \leq i \leq |A(v)|$, $\mathrm{dom}(L(v))_i = \begin{cases} \mathrm{cod}(L(w))_j & \text{if} \quad A_i(v) = \langle w, j\rangle \\ \sigma_j & \text{if} \quad A_i(v) = d_j \end{cases}$
  - If $c_i = \langle v, j\rangle$, $\mathrm{cod}(L(v))_j = \tau_i$. If $c_i = d_j$, $\sigma_j = \tau_i$.

$\square$

Example 2.1.5 Consider a sort $S = \{\mathtt{nat}\}$ and a signature $\Sigma = \{\mathtt{zero} : () \to (\mathtt{nat}), \mathtt{plus} : (\mathtt{nat}, \mathtt{nat}) \to (\mathtt{nat})\}$. Let us construct sharing graphs of type $() \to (\mathtt{nat})$ as drawn in the pictures below.

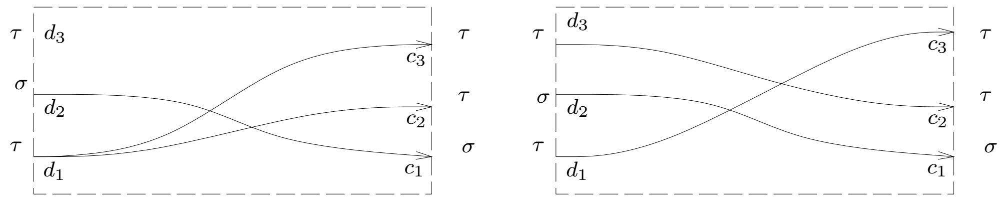
[1] For sets S and S', we write S^* for the set of finite lists of elements of S, and $S \oplus S'$ for the disjoint union of S and S'.

[2] Instead of $\{d_1, \ldots, d_m\}$, we can simply use natural numbers $\{1, \ldots, m\}$ – we did not do so just for the readability.



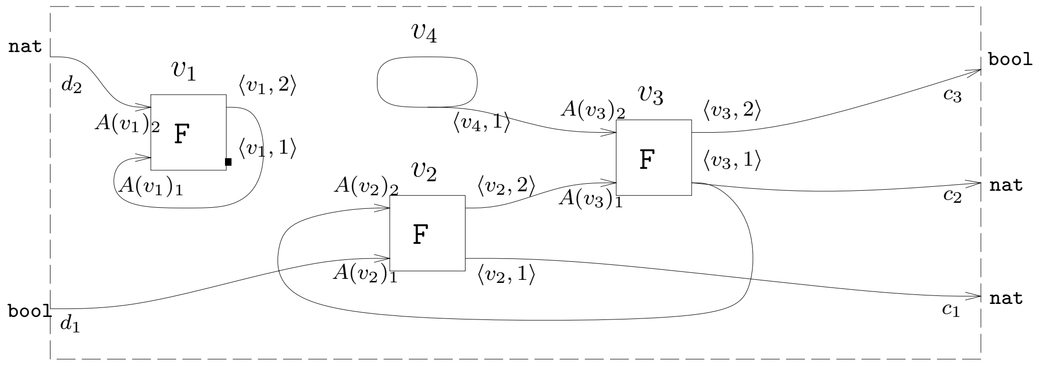
A sharing graph $(V, L, A, (c)) : () \to (\texttt{nat})$ for the left picture may be specified by $V = \{v, w\}$, $L(v) = \texttt{zero}$, $L(w) = \texttt{plus}$, $A(w)_1 = A(w)_2 = \langle v, 1\rangle$, and $c = \langle w, 1\rangle$. For the right one, V, L and c are unchanged but we modify A as $A(w)_1 = \langle v, 1\rangle$ and $A(w)_2 = \langle w, 1\rangle$. □

Example 2.1.6 If V is empty, A and L are the unique functions from the empty set, while $C = \{d_1, \ldots, d_m\}$. Thus such a sharing graph is determined by a function from $\{1, \ldots n\}$ to $\{1, \ldots, m\}$ subject to the type constraint. Below are two examples of such sharing graphs of type $(\tau, \sigma, \tau) \to (\sigma, \tau, \tau)$.



The left graph is specified by output nodes $c_1 = d_2$, $c_2 = d_1$ and $c_3 = d_1$. Similarly, the right one is specified as $c_1 = d_2$, $c_2 = d_1$ and $c_3 = d_1$. □

Example 2.1.7 A more tangled example. Let $S = \{\texttt{bool}, \texttt{nat}\}$ and $\Sigma = \{\texttt{F} : (\texttt{bool}, \texttt{nat}) \to (\texttt{nat}, \texttt{bool})\}$. A sharing graph $(V, L, A, (c_1, c_2, c_3))$ of type $(\texttt{bool}, \texttt{nat}) \to (\texttt{nat}, \texttt{nat}, \texttt{bool})$ may be given as below.
$V = \{v_1, v_2, v_3, v_4\}$.
$L(v_1) = L(v_2) = L(v_3) = \texttt{F}$, $L(v_4) = \bullet$.
$A(v_1)_1 = \langle v_1, 2\rangle$, $A(v_1)_2 = d_2$, $A(v_2)_1 = d_1$, $A(v_2)_2 = \langle v_3, 1\rangle$, $A(v_3)_1 = \langle v_2, 2\rangle$ and $A(v_3)_2 = \langle v_4, 1\rangle$.
$c_1 = \langle v_2, 1\rangle$, $c_2 = \langle v_3, 1\rangle$ and $c_3 = \langle v_3, 2\rangle$.



Note that v_4 ($\bullet^{\texttt{nat}}$) is drawn as a trivial cycle. $\Box$

Remark 2.1.8 The direction of the links drawn in the pictures is opposite to that of most of the pictorial presentations of sharing graphs in the literature. The reason we invert the direction will become clear when we introduce the categorical semantics of sharing graphs: our direction is that of morphisms in our semantic categories. $\Box$

Remark 2.1.9

- If we just consider operator symbols with just one output sort, then in the definition above C becomes $V \cup \{d_1, \ldots, d_m\}$ and we recover a standard definition like in [AK96] (c.f. [AB97]).
- In a standard terminology, assuming m inputs amounts to assuming m free variables $\{d_1, \ldots, d_m\}$.
- There is no technical difficulty to formulate the *infinitary* version of sharing graphs, by allowing infinitely many inputs and outputs (and allowing operators with infinitely many inputs and outputs too). However we do not see any practical benefit of such extra generality for our study (though infinitary operators must be included in some related settings, e.g. systems of equations for non-well-founded set theory [Acz88, BM96] which is out of the scope of this thesis); and giving the semantic models of the infinitary setting is far more complicated than the case for the finitary one, so we do not consider such a version. Also in this thesis we are mainly interested in the finite sharing graphs (see below), where the infinitary version seems meaningless. $\Box$

Definition 2.1.10 (finite sharing graph)
A sharing graph $(V, L, A, (c_1, \ldots, c_n))$ is *finite* if $|V| < \infty$. $\Box$

Definition 2.1.11 (equivalence of sharing graphs)
Two sharing graphs $G = (V, L, A, (c_1, \ldots, c_n))$, $G' = (V', L', A', (c'_1, \ldots, c'_n))$ of the same type are said to be *equivalent* if there is a bijection on sets $f : V \overset{\sim}{\rightarrow} V'$ such that $L' \circ f = L$, $A' \circ f = (f \times id)^* \circ A$ and $(f \times id)(c_i) = c'_i$. Obviously this determines an equivalence relation on sharing graphs of the same type, and we write $G \equiv G'$ if G and G' are equivalent. $\Box$

For instance, it is easily seen that two sharing graphs in Example 2.1.5 are not equivalent.

Definition 2.1.12 (dependency relation)
Let $(V, L, A, (c_1, \dots, c_n))$ be a sharing graph. We define a binary relation $<$ on V by $v < w$ if $A(w)_i = \langle v, j \rangle$ for some i, j, and also $v < v$ if $L(v) = \bullet$. Then the *dependency relation* $<^*$ is the transitive closure of $<$. $\Box$

For instance, in Example 2.1.7 the dependency relation consists of $v_1 <^* v_1$, $v_2 <^* v_2$, $v_2 <^* v_3$, $v_3 <^* v_2$, $v_3 <^* v_3$, $v_4 <^* v_3$ and $v_4 <^* v_4$.

Definition 2.1.13 (acyclic sharing graph)
A sharing graph $(V, L, A, (c_1, \dots, c_n))$ is *acyclic* if there is no $v \in V$ such that $v <^* v$. $\Box$

For example, one may see that the left graph in Example 2.1.5 is acyclic while the right is not (i.e. truly cyclic).

In the sequel, by a sharing graph, we may mean the equivalence class of the sharing graph (if there is no confusion). And we work just on finite sharing graphs unless explicitly mentioned.

2.2 Acyclic Sharing Theory

In this section we give the equational presentation of acyclic sharing graphs (cyclic graphs are dealt with in the following section). As discussed informally in the introduction, our presentation is based on "systems of equations", represented by the let-bindings in the acyclic case. So the expressions are terms of traditional algebraic theories plus let-blocks, and we assume a set of axioms which ensures that two expressions correspond to the same sharing graph if and only if they are provably equal in the theory. Though we need some care for dealing with multiple outputs (represented by tensor products), our development is fairly close to the standard stories for (many sorted) algebraic type theories as found in [Cro93] and we hope that our syntax is not very far from such traditional treatments. The comparison between sharing graphs and the equivalence classes of the terms of sharing theories is done by giving the translation between them, using a normal form property of the theories.

In the rest of this chapter, we fix a set of sorts S and a signature Σ, as introduced in the last section, unless otherwise stated.

Definition 2.2.1 (raw expressions)

$$M \ ::= \ x \mid F(M) \mid 0 \mid M_1 \otimes M_2 \mid \mathsf{let}\ (x_1, \ldots, x_m)\ \mathsf{be}\ M_1\ \mathsf{in}\ M_2$$

We assume that 0 and $\otimes$ satisfy *strict associativity*: we identify $0 \otimes M$, $M \otimes 0$ with M, and $(M_1 \otimes M_2) \otimes M_3$ with $M_1 \otimes (M_2 \otimes M_3)$ (and write $M_1 \otimes M_2 \otimes M_3$ for it). $\square$

The notion of free and bound variables is defined as usual, and we write $FV(M)$ and $BV(M)$ for the sets of free and bound variables of the expression M respectively. We write $M\{N/x\}$ for a capture-free substitution of N into free x's in M. Similarly we use $M\{N_1/x_1, N_2/x_2\}$, $M\{\vec{N}/\vec{x}\}$ etc for simultaneous substitutions. In the sequel, $\mathsf{let}\ (x_1, \ldots, x_m)\ \mathsf{be}\ M_1\ \mathsf{in}\ M_2$ may be written as $\mathsf{let}\ (\vec{x})\ \mathsf{be}\ M_1\ \mathsf{in}\ M_2$ for short. Also we may write $\vec{x}$ instead of $x_1 \otimes \ldots \otimes x_m$, which includes 0 as the case of $m = 0$.

Definition 2.2.2 (typing)
In the following, by a *context* we mean a finite list of pairs of variables and sorts like $x_1 : \sigma_1, \ldots, x_m : \sigma_m$ where x_i are different to each other. We say a term M has a type $(\sigma_1, \ldots, \sigma_n)$ under a context Γ if $\Gamma \vdash M : (\sigma_1, \ldots, \sigma_n)$ is derivable from the following typing rules. Such a term is called a *well-typed term*.

$$\frac{}{\Gamma, x : \sigma \vdash x : (\sigma)}\ \text{variable}$$

$$\frac{\Gamma \vdash M : (\sigma_1, \ldots, \sigma_m) \quad F : (\sigma_1, \ldots, \sigma_m) \to (\tau_1, \ldots, \tau_n)}{\Gamma \vdash F(M) : (\tau_1, \ldots, \tau_n)}\ \text{operator}$$

$$\frac{}{\Gamma \vdash 0 : ()}\ \text{unit}$$

$$\frac{\Gamma \vdash M : (\sigma_1, \ldots, \sigma_m) \quad \Gamma \vdash N : (\tau_1, \ldots, \tau_n)}{\Gamma \vdash M \otimes N : (\sigma_1, \ldots, \sigma_m, \tau_1, \ldots, \tau_n)}\ \text{tensor}$$

$$\frac{\Gamma \vdash M : (\sigma_1, \ldots, \sigma_m) \quad \Gamma, x_1 : \sigma_1, \ldots, x_m : \sigma_m \vdash N : (\tau_1, \ldots, \tau_n)}{\Gamma \vdash \mathsf{let}\ (x_1, \ldots, x_m)\ \mathsf{be}\ M\ \mathsf{in}\ N : (\tau_1, \ldots, \tau_n)}\ \text{let}$$

$$\frac{\Gamma, x : \sigma, x' : \sigma', \Gamma' \vdash M : (\tau_1, \ldots, \tau_n)}{\Gamma, x' : \sigma', x : \sigma, \Gamma' \vdash M : (\tau_1, \ldots, \tau_n)}\ \text{exchange}$$

$\square$

On notation: a list of sorts $(\sigma_1, \ldots, \sigma_m)$ may be abbreviated to $(\vec{\sigma})$.

Definition 2.2.3 (axioms)

$$
\begin{array}{lll}
(\sigma_{\mathrm{var}}) & \mathsf{let}\ (x)\ \mathsf{be}\ y\ \mathsf{in}\ M & = M\{y/x\} \\
(\mathrm{id}) & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{x} & = M \\
(\mathrm{ass}_1) & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ (\mathsf{let}\ (\vec{y})\ \mathsf{be}\ L\ \mathsf{in}\ M)\ \mathsf{in}\ N & = \mathsf{let}\ (\vec{y})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N \\
(\mathrm{ass}_2) & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N & = \mathsf{let}\ (\vec{x},\vec{y})\ \mathsf{be}\ L\otimes M\ \mathsf{in}\ N \\
(\otimes_1) & L\otimes(\mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N) & = \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ L\otimes N \\
(\otimes_2) & (\mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M)\otimes N & = \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M\otimes N \\
(\mathrm{subst}) & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ F(N) & = F(\mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N)
\end{array}
$$

They are "equations in contexts"; For each axiom, under the same context, the left hand side must have the same type with the right hand side. □

For instance, in (ass_2), x's cannot be free in M.

Definition 2.2.4 (acyclic sharing theory)
An *acyclic sharing theory* over Σ is an equational theory on the well-typed terms closed under the term construction described above, where the equality on terms is a congruence relation containing the axioms above, i.e. equations derivable from the following inference rules, possibly with additional axioms.

$$\frac{\Gamma\vdash M:(\vec{\tau})\quad \Gamma\vdash N:(\vec{\tau})\quad M=N \text{ is included in the axioms}}{\Gamma\vdash M=N:(\vec{\tau})}$$

$$\frac{\Gamma\vdash M:(\vec{\tau})}{\Gamma\vdash M=M:(\vec{\tau})}\qquad \frac{\Gamma\vdash M=N:(\vec{\tau})}{\Gamma\vdash N=M:(\vec{\tau})}\qquad \frac{\Gamma\vdash L=M:(\vec{\tau})\quad \Gamma\vdash M=N:(\vec{\tau})}{\Gamma\vdash L=N:(\vec{\tau})}$$

$$\frac{\Gamma\vdash M=N:(\vec{\sigma})\quad F:(\vec{\sigma})\to(\vec{\tau})}{\Gamma\vdash F(M)=F(N):(\vec{\tau})}\qquad \frac{\Gamma\vdash M=M':(\vec{\sigma})\quad \Gamma\vdash N=N':(\vec{\tau})}{\Gamma\vdash M\otimes N=M'\otimes N':(\vec{\sigma},\vec{\tau})}$$

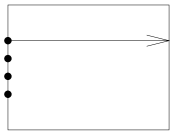
$$\frac{\Gamma\vdash M=M':(\vec{\sigma})\quad \Gamma,\vec{x}:\vec{\sigma}\vdash N=N':(\vec{\tau})}{\Gamma\vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N=\mathsf{let}\ (\vec{x})\ \mathsf{be}\ M'\ \mathsf{in}\ N':(\vec{\tau})}$$

$$\frac{\Gamma,x:\sigma,x':\sigma',\Gamma'\vdash M=N:(\vec{\tau})}{\Gamma,x':\sigma',x:\sigma,\Gamma'\vdash M=N:(\vec{\tau})}$$

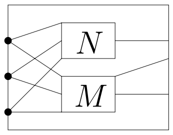
By the *pure acyclic sharing theory*, we mean the acyclic sharing theory with no additional axioms. □

In the sequel, the word "acyclic" may be dropped, unless there can be a confusion with the cyclic sharing theories which will be introduced later.

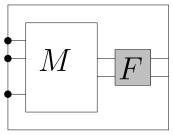
To help with the intuition, we give a pictorial account for our term constructions:



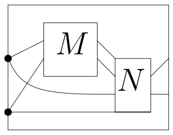
$$\frac{}{\Gamma, x : \sigma \vdash x : (\sigma)}$$



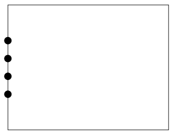
$$\frac{\Gamma \vdash M : (\vec{\sigma}) \quad \Gamma \vdash N : (\vec{\tau})}{\Gamma \vdash M \otimes N : (\vec{\sigma}, \vec{\tau})}$$



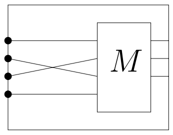
$$\frac{\Gamma \vdash M : (\vec{\sigma})}{\Gamma \vdash F(M) : (\vec{\tau})}$$



$$\frac{\Gamma \vdash M : (\vec{\sigma}) \quad \Gamma, \vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})}{\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})}$$



$$\frac{}{\Gamma \vdash 0 : ()}$$



$$\frac{\Gamma, x : \sigma, x' : \sigma', \Gamma' \vdash M : (\vec{\tau})}{\Gamma, x' : \sigma', x : \sigma, \Gamma' \vdash M : (\vec{\tau})}$$

Each term construction (type derivation) amounts to constructing a new graph from existing ones.

Remark 2.2.5 It might be helpful to understand the let-bindings as "suspended (or delayed) substitutions"; they perform the real substitution only for the case of variables (σ_{var}). If we write $N[M/\vec{x}]$ for $\mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N$, the axioms can be rewritten as

$$\boxed{\begin{array}{llcl}
(\sigma_{\text{var}}) & M[y/x] & = & M\{y/x\} \\
(\text{id}) & \vec{x}[M/\vec{x}] & = & M \\
(\text{ass}_1) & N[(M[L/\vec{y}])/\vec{x}] & = & (N[M/\vec{x}])[L/\vec{y}] \\
(\text{ass}_2) & (N[M/\vec{x}])[L/\vec{y}] & = & N[L \otimes M/\vec{x}, \vec{y}] \\
(\otimes_1) & L \otimes (N[M/\vec{x}]) & = & (L \otimes N)[M/\vec{x}] \\
(\otimes_2) & (M[L/\vec{x}]) \otimes N & = & (M \otimes N)[L/\vec{x}] \\
(\text{subst}) & (F(N))[M/\vec{x}] & = & F(N[M/\vec{x}])
\end{array}}$$

For instance, (ass_1) represents the associativity of substitutions. Other axioms are read similarly as specifying the expected properties of (suspended) substitutions. □

Interestingly, α-conversion of the let-binding is derivable from our axioms.

Lemma 2.2.6 (substitution of multiple variables and α-conversion)

1. If $\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})$ and $\vec{y}$ do not occur in Γ,

$$\Gamma, \vec{y} : \vec{\sigma}_m \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ \vec{y}\ \mathsf{in}\ M \ = \ M\{\vec{y}/\vec{x}\} : (\vec{\tau}).$$

2. If $\Gamma \vdash M : (\vec{\sigma})$, $\Gamma, \vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})$ and $\vec{y}$ do not occur in Γ nor in $\vec{x}$,

$$(\alpha) \quad \Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N \ = \ \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N\{\vec{y}/\vec{x}\} : (\vec{\tau}).$$

Proof:

1. The case of the number of variables m is 0:

$$\begin{array}{lll} & \mathsf{let}\ ()\ \mathsf{be}\ 0\ \mathsf{in}\ M & \\ = & \mathsf{let}\ ()\ \mathsf{be}\ 0\ \mathsf{in}\ \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{x} & (\mathrm{id}) \\ = & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{x} & (\mathrm{ass}_2) \\ = & M & (\mathrm{id}) \end{array}$$

For $m \geq 1$, by induction on the number of variables:

$$\begin{array}{lll} & \mathsf{let}\ (x_1, \ldots, x_{m+1})\ \mathsf{be}\ y_1 \otimes \ldots y_{m+1}\ \mathsf{in}\ M & \\ = & \mathsf{let}\ (x_1)\ \mathsf{be}\ y_1\ \mathsf{in}\ \mathsf{let}\ (x_2, \ldots, x_{m+1})\ \mathsf{be}\ y_2 \otimes \ldots y_{m+1}\ \mathsf{in}\ M & (\mathrm{ass}_2) \\ = & \mathsf{let}\ (x_1)\ \mathsf{be}\ y_1\ \mathsf{in}\ M\{y_2/x_2, \ldots, y_{m+1}/x_{m+1}\} & (\mathrm{hyp.}) \\ = & M\{y_1/x_1, \ldots, y_{m+1}/x_{m+1}\} & (\sigma_{\mathrm{var}}). \end{array}$$

2.

$$\begin{array}{lll} & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N & \\ = & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ (\mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{y})\ \mathsf{in}\ N & (\mathrm{id}) \\ = & \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ (\vec{x})\ \mathsf{be}\ \vec{y}\ \mathsf{in}\ N & (\mathrm{ass}_1) \\ = & \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N\{\vec{y}/\vec{x}\} & (1.). \quad \square \end{array}$$

Permutations of "independent" let-bindings are also justified:

Lemma 2.2.7 (permutation of let-bindings)
For $\Gamma \vdash L : (\vec{\sigma}_1)$, $\Gamma \vdash M : (\vec{\sigma}_2)$ and $\Gamma, \vec{x} : \vec{\sigma}_1, \vec{y} : \vec{\sigma}_2 \vdash N : (\vec{\tau})$,

$$\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N \ = \ \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ N : (\vec{\tau})$$

is derivable; we shall call this equation (perm).

Proof:

$$\begin{array}{lll} & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N & \\ = & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ (\vec{u}, \vec{v})\ \mathsf{be}\ \vec{x} \otimes \vec{y}\ \mathsf{in}\ N\{\vec{u}/\vec{x}, \vec{v}/\vec{y}\} & (\sigma_{\mathrm{var}}) \\ = & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ (\vec{u}, \vec{v})\ \mathsf{be}\ (\mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{x} \otimes \vec{y})\ \mathsf{in}\ N\{\vec{u}/\vec{x}, \vec{v}/\vec{y}\} & (\mathrm{ass}_1) \\ = & \mathsf{let}\ (\vec{u}, \vec{v})\ \mathsf{be}\ (\mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{x} \otimes \vec{y})\ \mathsf{in}\ N\{\vec{u}/\vec{x}, \vec{v}/\vec{y}\} & (\mathrm{ass}_1) \\ = & \mathsf{let}\ (\vec{u}, \vec{v})\ \mathsf{be}\ (\mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \vec{x} \otimes (\mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{y}))\ \mathsf{in}\ N\{\vec{u}/\vec{x}, \vec{v}/\vec{y}\} & (\otimes_2) \\ = & \mathsf{let}\ (\vec{u}, \vec{v})\ \mathsf{be}\ ((\mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \vec{x}) \otimes (\mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{y}))\ \mathsf{in}\ N\{\vec{u}/\vec{x}, \vec{v}/\vec{y}\} & (\otimes_1) \\ = & \mathsf{let}\ (\vec{u}, \vec{v})\ \mathsf{be}\ (\mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ (\mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \vec{x}) \otimes \vec{y})\ \mathsf{in}\ N\{\vec{u}/\vec{x}, \vec{v}/\vec{y}\} & (\otimes_2) \\ = & \mathsf{let}\ (\vec{u}, \vec{v})\ \mathsf{be}\ (\mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \vec{x} \otimes \vec{y})\ \mathsf{in}\ N\{\vec{u}/\vec{x}, \vec{v}/\vec{y}\} & (\otimes_1) \\ = & \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ (\vec{u}, \vec{v})\ \mathsf{be}\ (\mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \vec{x} \otimes \vec{y})\ \mathsf{in}\ N\{\vec{u}/\vec{x}, \vec{v}/\vec{y}\} & (\mathrm{ass}_1) \\ = & \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ (\vec{u}, \vec{v})\ \mathsf{be}\ \vec{x} \otimes \vec{y}\ \mathsf{in}\ N\{\vec{u}/\vec{x}, \vec{v}/\vec{y}\} & (\mathrm{ass}_1) \\ = & \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ N & (\sigma_{\mathrm{var}}) \end{array}$$

$\square$

To help with the intuition, we shall look at a few examples.

Example 2.2.8 As Example 2.1.5, we consider a sort $S = \{\texttt{nat}\}$ and a signature $\Sigma = \{\texttt{zero} : () \to (\texttt{nat}), \texttt{plus} : (\texttt{nat}, \texttt{nat}) \to (\texttt{nat})\}$. In an acyclic sharing theory over Σ, we can present the first sharing graph in Example 2.1.5 as

$$\vdash \mathsf{let}\ (x)\ \mathsf{be}\ \texttt{zero}\ \mathsf{in}\ \texttt{plus}(x \otimes x) : (\texttt{nat}).$$

It is easy to see that, in the pure acyclic sharing theory, this term is not equivalent to the "unshared" version

$$\vdash \texttt{plus}(\texttt{zero} \otimes \texttt{zero}) : (\texttt{nat})$$

because each axiom preserves the number of occurrences of operator symbols. However, as acyclic sharing theories lack the ability of representing cyclic bindings, we cannot present the second cyclic graph. □

Example 2.2.9 In the example above, the terms can be expressed in many different ways. For instance,

$$\begin{array}{cll}
 & \texttt{plus}(\texttt{zero} \otimes \texttt{zero}) & \\
= & \mathsf{let}\ (x, y)\ \mathsf{be}\ \texttt{zero} \otimes \texttt{zero}\ \mathsf{in}\ \texttt{plus}(x \otimes y) & (\text{subst}) \\
= & \mathsf{let}\ (x)\ \mathsf{be}\ \texttt{zero}\ \mathsf{in}\ \mathsf{let}\ (y)\ \mathsf{be}\ \texttt{zero}\ \mathsf{in}\ \texttt{plus}(x \otimes y) & (\text{ass}_2) \\
= & \mathsf{let}\ (x)\ \mathsf{be}\ \texttt{zero}\ \mathsf{in}\ \texttt{plus}(\mathsf{let}\ (y)\ \mathsf{be}\ \texttt{zero}\ \mathsf{in}\ x \otimes y) & (\text{subst}) \\
= & \mathsf{let}\ (x)\ \mathsf{be}\ \texttt{zero}\ \mathsf{in}\ \texttt{plus}(x \otimes (\mathsf{let}\ (y)\ \mathsf{be}\ \texttt{zero}\ \mathsf{in}\ x \otimes y)) & (\otimes_1) \\
= & \mathsf{let}\ (x)\ \mathsf{be}\ \texttt{zero}\ \mathsf{in}\ \texttt{plus}(x \otimes \texttt{zero}) & (\text{id})
\end{array}$$

and so on; we note that the third line is a normal form of this term (see below). □

Example 2.2.10 One may wish to ignore isolated resources, for instance wanting to equate $\mathsf{let}\ (x)\ \mathsf{be}\ \texttt{zero}\ \mathsf{in}\ M$ with M if x is not free in M (thus the resource $\texttt{zero}$ is not referred from anywhere). This is not derivable in the pure acyclic sharing theory, and we need to assume an additional axiom (*garbage collection*):

$$M = 0 \quad \text{for any } \Gamma \vdash M : ()$$

Then, for instance, one can derive

$$\begin{array}{cll}
 & \mathsf{let}\ (x)\ \mathsf{be}\ \texttt{zero}\ \mathsf{in}\ M & \\
= & \mathsf{let}\ ()\ \mathsf{be}\ (\mathsf{let}\ (x)\ \mathsf{be}\ \texttt{zero}\ \mathsf{in}\ 0)\ \mathsf{in}\ M & \text{ass}_1 \\
= & \mathsf{let}\ ()\ \mathsf{be}\ 0\ \mathsf{in}\ M & \text{garbage collection} \\
= & M. &
\end{array}$$

(Semantically, this condition amounts to assuming that () is a terminal object. It is also called the affineness condition in the literature, c.f. [Jac94, MOTW95].) □

Remark 2.2.11 Note that we have "empty bindings" for terms of type $()$, like $\mathsf{let}\ ()\ \mathsf{be}\ M\ \mathsf{in}\ N$ where s is of type $()$, as the case of the list of binding variables is empty. Actually there is no real binding of the subterm M in this expression; M can move around everywhere in the whole term.

$$\begin{array}{rcll} \mathsf{let}\ ()\ \mathsf{be}\ M\ \mathsf{in}\ N & = & \mathsf{let}\ ()\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ (\vec{x})\ \mathsf{be}\ N\ \mathsf{in}\ \vec{x} & (\mathrm{id}) \\ & = & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M \otimes N\ \mathsf{in}\ \vec{x} & (\mathrm{ass}_2) \\ & = & M \otimes N & (\mathrm{id}) \end{array}$$

Using (perm), one can prove that $\mathsf{let}\ ()\ \mathsf{be}\ M\ \mathsf{in}\ N = M \otimes N = N \otimes M$ for any $M : ()$. In particular, (the equivalence classes of) terms of type $()$ form a commutative monoid. □

Definition 2.2.12 (normal forms)
Normal forms are the well-typed terms generated from the following grammar.

$$n \ ::= \ \vec{x} \ \mid \ \mathsf{let}\ (\vec{x})\ \mathsf{be}\ F(\vec{y})\ \mathsf{in}\ n$$

□

Theorem 2.2.13 (normal form theorem)
In the pure sharing theory, for any term $\Gamma \vdash M : (\vec{\tau})$, there is a normal form $\Gamma \vdash n : (\vec{\tau})$ such that $M = n$. Such a normal form is unique up to the congruence generated by (α) and (perm).

Proof: For existence, one may start from proving that if n, n' are normal forms $\mathsf{let}\ (\vec{x})\ \mathsf{be}\ n\ \mathsf{in}\ n'$ is equal to a normal form by the induction on the construction of n, taking care that substitution of variables preserves normal forms. Uniqueness follows from the fact that any axiom preserves operator symbols. □

Remark 2.2.14 Our non-standard syntax arises from the existence of multiple conclusions (outputs). If we are interested just in single conclusions, then the following more standard syntax suffices.

$$M \ ::= \ x \mid F(M_1, \ldots, M_m) \mid \mathsf{let}\ x\ \mathsf{be}\ M\ \mathsf{in}\ N$$

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ \text{variable}$$

$$\frac{\Gamma \vdash M_i : \sigma_i \quad (1 \leq i \leq m) \quad F : (\sigma_1, \ldots, \sigma_m) \to \tau}{\Gamma \vdash F(M_1, \ldots, M_m) : \tau}\ \text{operator}$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma, x : \sigma \vdash N : \tau}{\Gamma \vdash \mathsf{let}\ x\ \mathsf{be}\ M\ \mathsf{in}\ N : \tau}\ \text{let}$$

$$
\begin{array}{lcl}
\mathsf{let}\ x\ \mathsf{be}\ y\ \mathsf{in}\ M & = & M\{y/x\} \\
\mathsf{let}\ x\ \mathsf{be}\ M\ \mathsf{in}\ x & = & M \\
\mathsf{let}\ x\ \mathsf{be}\ (\mathsf{let}\ y\ \mathsf{be}\ L\ \mathsf{in}\ M)\ \mathsf{in}\ N & = & \mathsf{let}\ y\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ x\ \mathsf{be}\ M\ \mathsf{in}\ N \\
\mathsf{let}\ x_1\ \mathsf{be}\ M_1\ \mathsf{in}\ \mathsf{let}\ x_2\ \mathsf{be}\ M_2\ \mathsf{in}\ N & = & \mathsf{let}\ x_2\ \mathsf{be}\ M_2\ \mathsf{in}\ \mathsf{let}\ x_1\ \mathsf{be}\ M_1\ \mathsf{in}\ N \\
\mathsf{let}\ x\ \mathsf{be}\ M\ \mathsf{in}\ F(\ldots,N,\ldots) & = & F(\ldots,\mathsf{let}\ x\ \mathsf{be}\ M\ \mathsf{in}\ N,\ldots)
\end{array}
$$

It is routine to check that all axioms above are derivable from those of the pure acyclic sharing theory, thus there is a sound interpretation of this restricted version into the bigger theory. Moreover it is also easy to define the normal forms for this version, which are properly included in those for the pure acyclic sharing theory. So we conclude that the pure acyclic sharing theory is conservative over this restricted theory. □

Relating Normal Forms and Sharing Graphs

We relate the pure acyclic sharing theory with finite acyclic sharing graphs as follows.

1. First, we show that acyclic graphs and normal forms are in bijective correspondence, up to renaming of bound variables (α-conversion) and permutations of operators which do not interfere with the dependency relation.

2. Since any term has a unique normal form up to α-conversion and (perm), we conclude that the acyclic sharing theory precisely axiomatizes finite acyclic sharing graphs.

Let $G = (V, L, A, (c_1, \ldots, c_n)) : (\sigma_1, \ldots, \sigma_m) \to (\tau_1, \ldots, \tau_n)$ be an acyclic sharing graph. Then it is possible to sort V's elements as v_1,...,v_k so that $i \leq j$ implies $v_j \not<^* v_i$. Of course this choice may not be unique, but we shall choose one of them, and build an expression of the pure acyclic sharing theory as

$$
\begin{array}{rcl}
M_G & \equiv & \mathsf{let}\ (x_{\langle v_1,1\rangle}, \ldots, x_{\langle v_1,|\mathrm{cod}(L(v_1))|\rangle})\ \mathsf{be}\ L(v_1)(x_{A(v_1)_1}, \ldots, x_{A(v_1)_{|\mathrm{dom}(L(v_1))|}})\ \mathsf{in} \\
& & \ldots \\
& & \mathsf{let}\ (x_{\langle v_k,1\rangle}, \ldots, x_{\langle v_k,|\mathrm{cod}(L(v_k))|\rangle})\ \mathsf{be}\ L(v_k)(x_{A(v_k)_1}, \ldots, x_{A(v_k)_{|\mathrm{dom}(L(v_k))|}})\ \mathsf{in} \\
& & x_{c_1} \otimes \ldots \otimes x_{c_n}
\end{array}
$$

where we introduce a variable x_c for each $c \in C$ (recall the definition). We claim that M_G is a well-typed expression (hence a normal form):

Lemma 2.2.15 $x_{d_1} : \sigma_1, \ldots, x_{d_m} : \sigma_m \vdash M_G : (\tau_1, \ldots, \tau_n)$.

Proof: Induction on the size of V. If V is empty, M_G is simply $x_{c_1} \otimes \ldots \otimes x_{c_n}$, and since each c_i is contained in $\{d_1, \ldots, d_m\}$ M_G's typing is derivable from the rules (variable) and (tensor) (if $n > 0$) or (unit) (if $n = 0$). If V is not empty, one may consider a sharing graph $G' = (V', A', L', (c_1, \ldots, c_n)) : (\sigma_1, \ldots, \sigma_{m+|\mathrm{dom}(L(v_1))|} \rightarrow (\tau_1, \ldots, \tau_n)$ where $V' = V - \{v_1\}$, $L' = L|_{V'}$, $A'(v)_i = d_{m+j}$ if $A'(v)_i = \langle v_1, j\rangle$ and $A'(v)_i = A(v)_i$ otherwise. σ_{m+i} is $\mathrm{dom}(L(v_1))_i$. Then (by the same linear-ordering)

$$\begin{array}{rcl} M_G & \equiv & \mathsf{let}\ (x_{\langle v_1,1\rangle}, \ldots, x_{\langle v_1,|\mathrm{cod}(L(v_1))|\rangle}) \\ & & \mathsf{be}\ L(v_1)(x_{A(v_1)_1}, \ldots, x_{A(v_1)_{|\mathrm{dom}(L(v_1))|}}) \\ & & \mathsf{in}\ M_{G'}\{x_{\langle v_1,j\rangle}/x_{d_{m+j}}\} \end{array}$$

On the other hand, by the induction hypothesis,

$$x_{d_1} : \sigma_1, \ldots, x_{d_{m+|\mathrm{dom}(L(v_1))}} : \sigma_{m+|\mathrm{dom}(L(v_1))|} \vdash M'_G : (\tau_1, \ldots, \tau_n)$$

By the rule (operator), (let) and renaming of variables by $x_{d_{m+j}} \mapsto x_{\langle v_1,j\rangle}$ we obtain the typing of M_G. $\Box$

By assuming another linear ordering of V's elements, we get another well-typed expression M'_G. However, the difference between M_G and M'_G lies only in the permutation of the let-bindings, which is guaranteed by (perm) (routinely shown by induction on the number of nodes). Thus $M_G = M'_G$ is provable in the sharing calculus. Also it is routine to check that equivalent sharing graphs give equivalent normal forms. Conversely, a normal form n gives rise to a sharing graph G_n, by reversing the construction above, such that $G_{M_G} \equiv G$ and $M_{G_n} = n$. Together with the proposition of normal forms, we have

Theorem 2.2.16 There is a bijective correspondence between the finite acyclic sharing graphs and the equivalence classes of terms of the pure acyclic sharing theory. $\Box$

Actually this should be more than a bijective correspondence; this must preserve the algebraic nature behind sharing graphs, though at this moment it is not very clear. The leading paradigm in this thesis is that "sharing graphs (and their models) form a nice categorical structure" – this bijection turns out to respect such semantic structure behind sharing graphs (see Remark 3.3.8). While not many work in the literature spot on the structural nature of sharing graphs, we shall point out that Milner's treatment of his "molecular forms" as an action structure (symmetric monoidal category with structure) [Mil96] certainly lies in this direction, and we acknowledge the influence of his work on ours.

2.3 Cyclic Sharing Theory

In a parallel manner to the acyclic sharing graphs and acyclic sharing theory, we give the equational theory for cyclic sharing graphs, called *cyclic sharing theory.* Ignoring small technical points, all we need is to replace the let-bindings (acyclic bindings) by the letrec-bindings (cyclic bindings). The syntax is close to that of many applicative programming languages with recursive bindings, although at this moment our theory corresponds to the cyclic sharing graphs and says nothing about (lazy) recursive computation as often represented by the letrec-syntax in lazy functional languages.

Definition 2.3.1 (raw expressions)

$$M \;::=\; x \mid F(M) \mid 0 \mid M_1 \otimes M_2 \mid \mathsf{letrec}\ (x_1, \ldots, x_m)\ \mathsf{be}\ M_1\ \mathsf{in}\ M_2$$

We identify $M \otimes 0$ and $0 \otimes M$ with M, and $(M_1 \otimes M_2) \otimes M_3$ with $M_1 \otimes (M_2 \otimes M_3)$.

Definition 2.3.2 (typing)

$$\frac{}{\Gamma, x : \sigma \vdash x : (\sigma)}\ \text{variable}$$

$$\frac{\Gamma \vdash M : (\sigma_1, \ldots, \sigma_m) \quad F : (\sigma_1, \ldots, \sigma_m) \to (\tau_1, \ldots, \tau_n)}{\Gamma \vdash F(M) : (\tau_1, \ldots, \tau_n)}\ \text{operator}$$

$$\frac{}{\Gamma \vdash 0 : ()}\ \text{unit}$$

$$\frac{\Gamma \vdash M : (\sigma_1, \ldots, \sigma_m) \quad \Gamma \vdash N : (\tau_1, \ldots, \tau_n)}{\Gamma \vdash M \otimes N : (\sigma_1, \ldots, \sigma_m, \tau_1, \ldots, \tau_n)}\ \text{tensor}$$

$$\frac{\begin{array}{c}\Gamma, x_1 : \sigma_1, \ldots, x_m : \sigma_m \vdash M : (\sigma_1, \ldots, \sigma_m) \\ \Gamma, x_1 : \sigma_1, \ldots, x_m : \sigma_m \vdash N : (\tau_1, \ldots, \tau_n)\end{array}}{\Gamma \vdash \mathsf{letrec}\ (x_1, \ldots, x_m)\ \mathsf{be}\ M\ \mathsf{in}\ N : (\tau_1, \ldots, \tau_n)}\ \text{letrec}$$

$$\frac{\Gamma, x : \sigma, x' : \sigma', \Gamma' \vdash M : (\tau_1, \ldots, \tau_n)}{\Gamma, x' : \sigma', x : \sigma, \Gamma' \vdash M : (\tau_1, \ldots, \tau_n)}\ \text{exchange}$$

□

Definition 2.3.3 (axioms)

(σ_{var})	$\mathsf{letrec}\ (x, \vec{y})\ \mathsf{be}\ z \otimes M\ \mathsf{in}\ N$	$= \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ M\{z/x\}\ \mathsf{in}\ N\{z/x\}$ $(x \not\equiv z)$
(id)	$\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{x}$	$= M$
(ass_1)	$\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ (\mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ L\ \mathsf{in}\ M)\ \mathsf{in}\ N$	$= \mathsf{letrec}\ (\vec{x}, \vec{y})\ \mathsf{be}\ M \otimes L\ \mathsf{in}\ N$
(ass_2)	$\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N$	$= \mathsf{letrec}\ (\vec{x}, \vec{y})\ \mathsf{be}\ L \otimes M\ \mathsf{in}\ N$
$(\otimes_1)$	$L \otimes (\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N)$	$= \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ L \otimes N$
$(\otimes_2)$	$(\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M) \otimes N$	$= \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M \otimes N$
(perm)	$\mathsf{letrec}\ (\vec{x}, \vec{y}, \vec{z})\ \mathsf{be}\ M_1 \otimes M_2 \otimes M_3\ \mathsf{in}\ N$	$= \mathsf{letrec}\ (\vec{y}, \vec{x}, \vec{z})\ \mathsf{be}\ M_2 \otimes M_1 \otimes M_3\ \mathsf{in}\ N$
(subst)	$\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ F(N)$	$= F(\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N)$

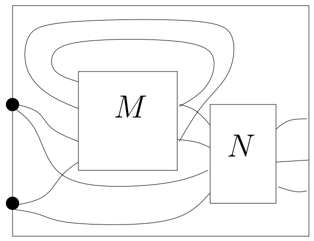
As the acyclic case, both sides of axioms must have the same type under the same context. For instance, in (id), $\vec{x}$ cannot be free in M. □

Definition 2.3.4 (cyclic sharing theory)
A *cyclic sharing theory* over Σ is an equational theory on the well-typed terms closed under the term construction described above, where the equality on terms is a congruence relation containing the axioms above; the only difference from that of an acyclic sharing theory is the following inference rule for letrec is used instead of that for let.

$$\frac{\Gamma, \vec{x} : \vec{\sigma} \vdash M = M' : (\vec{\sigma}) \quad \Gamma, \vec{x} : \vec{\sigma} \vdash N = N' : (\vec{\tau})}{\Gamma \vdash \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N = \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M'\ \mathsf{in}\ N' : (\vec{\tau})}$$

By the *pure cyclic sharing theory*, we mean the cyclic sharing theory with no additional axioms. □

Again we give a pictorial account for the letrec-construction:



$$\frac{\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\sigma}) \quad \Gamma, \vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})}{\Gamma \vdash \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})}$$

For readability, we introduce the following syntax for the multiple letrec-binding:

$$\frac{\begin{array}{c}\Gamma, \vec{x}_1 : \vec{\sigma}_1, \ldots, \vec{x}_k : \vec{\sigma}_k \vdash M_i : (\vec{\sigma}_i) \quad (1 \leq i \leq k) \\ \Gamma, \vec{x}_1 : \vec{\sigma}_1, \ldots, \vec{x}_k : \vec{\sigma}_k \vdash N : (\vec{\tau})\end{array}}{\Gamma \vdash \mathsf{letrec}\ (\vec{x}_1)\ \mathsf{be}\ M_1, \ldots\ (\vec{x}_k)\ \mathsf{be}\ M_k\ \mathsf{in}\ N : (\vec{\tau})}$$

for $\mathsf{letrec}\ (\vec{x}_1, \ldots, \vec{x}_k)\ \mathsf{be}\ M_1 \otimes \ldots \otimes M_k\ \mathsf{in}\ N$. For instance, the axiom (perm) is equivalent to

$$\mathsf{letrec}\ \ldots\ (\vec{x})\ \mathsf{be}\ M,\ (\vec{y})\ \mathsf{be}\ M'\ \ldots\ \mathsf{in}\ N\ =\ \mathsf{letrec}\ \ldots\ (\vec{y})\ \mathsf{be}\ M',\ (\vec{x})\ \mathsf{be}\ M\ \ldots\ \mathsf{in}\ N.$$

which will be referred as (perm'). Also, the axiom (σ_{var}) is the same as

$$\mathsf{letrec}\ (x)\ \mathsf{be}\ z,\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N\ =\ \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ M\{z/x\}\ \mathsf{in}\ N\{z/x\}$$

where $x \not\equiv z$. It is easy but helpful to see that this can be replaced by a pair of axioms representing "dereference" and "garbage collection" (restricted on variables):

$$\begin{array}{lll}
\text{(deref)} & \mathsf{letrec}\ (x)\ \mathsf{be}\ z,\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N & = \mathsf{letrec}\ (x)\ \mathsf{be}\ z,\ (\vec{y})\ \mathsf{be}\ M\{z/x\}\ \mathsf{in}\ N\{z/x\} \\
\text{(g.c.)} & \mathsf{letrec}\ (x)\ \mathsf{be}\ z\ \mathsf{in}\ N & = N \quad (x \notin FV(N)\ \&\ x \not\equiv z)
\end{array}$$

Note that the garbage collection of trivial cycles is not allowed (in the pure cyclic sharing theory), e.g. $\mathsf{letrec}\ (x)\ \mathsf{be}\ x\ \mathsf{in}\ M \ = \ M$ is not derivable even when x is not free in M. This alternative axiomatization will be used when we consider the extension with higher-order constructs where substitutions of values are allowed.

As in the acyclic theory, α-conversion of letrec-bound variables is derivable.

Lemma 2.3.5 (α-conversion)
If $\Gamma \vdash M : (\vec{\sigma})$, $\Gamma, \vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})$ and $\vec{y}$ do not occur in Γ nor in $\vec{x}$,

$$(\alpha) \quad \Gamma \vdash \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N \ = \ \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ M\{\vec{y}/\vec{x}\}\ \mathsf{in}\ N\{\vec{y}/\vec{x}\} : (\vec{\tau}).$$

Proof:

$$\begin{array}{lll}
 & \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N & \\
= & \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ (\mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{y})\ \mathsf{in}\ N & \text{(id)} \\
= & \mathsf{letrec}\ (\vec{x}, \vec{y})\ \mathsf{be}\ \vec{y} \otimes M\ \mathsf{in}\ N & (\text{ass}_1) \\
= & \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ M\{\vec{y}/\vec{x}\}\ \mathsf{in}\ N\{\vec{y}/\vec{x}\} & (\sigma_{\text{var}}).
\end{array}$$

□

The difference between the axiomatizations of acyclic sharing theory and the cyclic one is summarized as follows.

- The axiom (σ_{var}) is changed to so that a substitution is effective throughout under the scope of the letrec-binding. The acyclic version is too weak, for instance for deriving $\mathsf{letrec}\ (x, y)\ \mathsf{be}\ y \otimes x\ \mathsf{in}\ y \ = \ \mathsf{letrec}\ (y)\ \mathsf{be}\ y\ \mathsf{in}\ y$. However, there is no other difference; z must be different from x, so we cannot use this axiom for deleting a trivial cycle as mentioned above.

- The axiom (perm) is derivable in the acyclic theory, but it is not the case for the cyclic theory, because of the cyclic bindings. In (perm), M_i's can depend on each other, i.e. can contain any free x, y and z's, so the proof of (perm) for the acyclic theory using (ass$_1$) and ($\otimes_i$) cannot be applied.

- The axiom (ass$_1$) is slightly more general than that for the acyclic theory for free xs can occur in L and M.

However, the other axioms (id), (ass$_2$), ($\otimes_i$) and (subst) are unchanged.

Example 2.3.6 The second graph of Example 2.1.5 can be presented as

$$\vdash \mathsf{letrec}\ (x)\ \mathsf{be}\ \mathtt{plus}(\mathtt{zero}, x)\ \mathsf{in}\ x : (\mathtt{nat}).$$

□

Example 2.3.7 The tangled graph of Example 2.1.7 can be presented as

$$\begin{array}{rlll} x : \mathtt{bool}, y : \mathtt{nat} \vdash & \mathsf{letrec} & (z, z')\ \mathsf{be}\ \mathtt{F}(z', y), & \\ & & (u, u')\ \mathsf{be}\ \mathtt{F}(x, v), & \\ & & (v, v')\ \mathsf{be}\ \mathtt{F}(u', w), & \\ & & (w)\ \mathsf{be}\ w & \\ & \mathsf{in} & u \otimes v \otimes v' & : \mathtt{nat} \otimes \mathtt{nat} \otimes \mathtt{bool}. \end{array}$$

□

For defining the notion of normal forms, we need a care about trivial cycles again.

Definition 2.3.8 For each sort σ, define $\bullet^\sigma : (\sigma)$ by $\bullet^\sigma \equiv \mathsf{letrec}\ (x)\ \mathsf{be}\ x\ \mathsf{in}\ x$.
□

Definition 2.3.9 (normal forms)
Normal forms are the well-typed terms of the following form

$$\mathsf{letrec}\ (\vec{x}_1)\ \mathsf{be}\ F_1(\vec{y}_1)\ \mathsf{in}\ \ldots\ (\vec{x}_m)\ \mathsf{be}\ F_m(\vec{y}_m)\ \mathsf{in}\ \vec{z}$$

where F is either an operator symbol or $\bullet$. □

Theorem 2.3.10 (normal form theorem)
In the pure cyclic sharing theory, for any term $\Gamma \vdash M : (\vec{\tau})$, there is a normal form $\Gamma \vdash n : (\vec{\tau})$ such that $M = n$. Such a normal form is unique up to the congruence generated by (α) and (perm').

Proof: Similar to the case of the pure acyclic sharing theory. (N.B. [Mil94b] contains the essentially same result on reflexive action calculi and molecular forms.)
□

It is easy to give a cyclic sharing graph from a normal form and vise versa – unlike the acyclic case, we do not have to care about the dependency between nodes, and it is routine to show

Theorem 2.3.11 There is a bijective correspondence between the finite cyclic sharing graphs and the equivalence classes of terms of the pure cyclic sharing theory. □

Again this should be more than a bijective correspondence; this must preserve the algebraic nature implicit in cyclic sharing graphs. This becomes the central issue in the later development of the semantic counterpart.

Remark 2.3.12 The same remark applies as in the acyclic case: if we are interested just in the single conclusion, then the following more standard syntax suffices.

$$\begin{array}{llll} \text{Raw Terms} & M & ::= & x \mid F(M_1,\ldots,M_m) \mid \mathsf{letrec}\ x_1\ \mathsf{be}\ M_1,\ldots,x_m\ \mathsf{be}\ M_m\ \mathsf{in}\ N \\ \text{Declarations} & D & ::= & x\ \mathsf{be}\ M \mid x\ \mathsf{be}\ M, D \end{array}$$

In a declaration, binding variables are assumed to be disjoint.

$$\frac{}{\Gamma, x:\sigma \vdash x:\sigma}\ \text{variable}$$

$$\frac{\Gamma \vdash M_i:\sigma_i \quad (1 \le i \le m) \quad F:(\sigma_1,\ldots,\sigma_m)\to\tau}{\Gamma \vdash F(M_1,\ldots,M_m):\tau}\ \text{operator}$$

$$\frac{\begin{array}{c}\Gamma, x_1:\sigma_1,\ldots,x_m:\sigma_m \vdash M_i:\sigma_i \quad (1\le i\le m)\\ \Gamma, x_1:\sigma_1,\ldots,x_m:\sigma_m \vdash N:\tau\end{array}}{\Gamma \vdash \mathsf{letrec}\ x_1\ \mathsf{be}\ M_1,\ldots,x_m\ \mathsf{be}\ M_m\ \mathsf{in}\ N:\tau}\ \text{letrec}$$

$$\boxed{\begin{array}{lcl} \mathsf{letrec}\ x\ \mathsf{be}\ z, D\ \mathsf{in}\ M & = & \mathsf{letrec}\ D\{z/x\}\ \mathsf{in}\, M\{z/x\} \\ & & (x \not\equiv z) \\ \mathsf{letrec}\ x\ \mathsf{be}\ M\ \mathsf{in}\ x & = & M \\ \mathsf{letrec}\ y\ \mathsf{be}\ (\mathsf{letrec}\ D_1\ \mathsf{in}\ M), D_2\ \mathsf{in}\ N & = & \mathsf{letrec}\ D_1, y\ \mathsf{be}\ M, D_2\ \mathsf{in}\ N \\ \mathsf{letrec}\ D_1\ \mathsf{in}\ \mathsf{letrec}\ D_2\ \mathsf{in}\ M & = & \mathsf{letrec}\ D_1, D_2\ \mathsf{in}\ M \\ \mathsf{letrec}\ D_1, D_2, D_3\ \mathsf{in}\ N & = & \mathsf{letrec}\ D_2, D_1, D_3\ \mathsf{in}\ N \\ \mathsf{letrec}\ x\ \mathsf{be}\ M\ \mathsf{in}\ F(\ldots,N,\ldots) & = & F(\ldots,\mathsf{letrec}\ x\ \mathsf{be}\ M\ \mathsf{in}\ N,\ldots) \end{array}}$$

□

2.4 Rewriting on Sharing Graphs

Rewriting is the computational counterpart of sharing graphs. There is a wide variety of formulations of rewriting on sharing graphs from abstract ones to concrete implementation oriented ones [SPvE93], but here we choose an intermediate approach, which is close to Ariola and Klop's *equational term graph rewriting*.

Definition 2.4.1 (rewriting system)
Let $\mathbb{T}$ be a (acyclic or cyclic) sharing theory. A *rewriting system on* $\mathbb{T}$ is a preorder on the equivalence classes of well-typed terms with the same contexts and same types, closed under all term constructions. □

Spelling this out, a rewriting system can be described as a family of relations $\succ_{\Gamma,(\vec{\sigma})}$ for each context Γ and type $(\vec{\sigma})$ satisfying the following conditions.

- $\succ_{\Gamma,(\vec{\sigma})}$ is a binary relation on well-typed terms with context Γ and type $(\vec{\sigma})$ (subscripts may be omitted). We write $\Gamma \vdash M \succ N : (\vec{\sigma})$ for $M \succ_{\Gamma,(\vec{\sigma})} N$.

- $\succ_{\Gamma,(\vec{\sigma})}$ respects equality:
$$\frac{\Gamma \vdash M = N : (\vec{\sigma})}{\Gamma \vdash M \succ N : (\vec{\sigma})}$$

- $\succ_{\Gamma,(\vec{\sigma})}$ is a preorder:
$$\frac{\Gamma \vdash M : (\vec{\sigma})}{\Gamma \vdash M \succ M : (\vec{\sigma})} \quad \frac{\Gamma \vdash L \succ M : (\vec{\sigma}) \quad \Gamma \vdash M \succ N : (\vec{\sigma})}{\Gamma \vdash L \succ N : (\vec{\sigma})}$$

- Closure under term constructions:
$$\frac{\Gamma \vdash M \succ N : (\vec{\sigma}) \quad F : (\vec{\sigma}) \to (\vec{\tau})}{\Gamma \vdash F(M) \succ F(N) : (\vec{\tau})} \quad \frac{\Gamma \vdash M \succ M' : (\vec{\sigma}) \quad \Gamma \vdash N \succ N' : (\vec{\tau})}{\Gamma \vdash M \otimes N \succ M' \otimes N' : (\vec{\sigma}, \vec{\tau})}$$
$$\frac{\Gamma, x : \sigma, x' : \sigma', \Gamma' \vdash M \succ N : (\vec{\tau})}{\Gamma, x' : \sigma', x : \sigma, \Gamma' \vdash M \succ N : (\vec{\tau})}$$
For acyclic theory
$$\frac{\Gamma \vdash M \succ M' : (\vec{\sigma}) \quad \Gamma, \vec{x} : \vec{\sigma} \vdash N \succ N' : (\vec{\tau})}{\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N \succ \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M'\ \mathsf{in}\ N' : (\vec{\tau})}$$
For cyclic theory
$$\frac{\Gamma, \vec{x} : \vec{\sigma} \vdash M \succ M' : (\vec{\sigma}) \quad \Gamma, \vec{x} : \vec{\sigma} \vdash N \succ N' : (\vec{\tau})}{\Gamma \vdash \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N \succ \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M'\ \mathsf{in}\ N' : (\vec{\tau})}$$

This notion of rewriting is too rough. In most practical settings, it is unlikely that some resource is created from nothing, but our definition allows such a stupid rewriting system! Thus we are lead to restrict the notion of rewriting in an adequate way:

Definition 2.4.2 (minimality condition)
A rewriting system satisfies the *minimality condition* if $\Gamma \vdash \vec{x} \succ M : (\vec{\sigma})$ implies $\Gamma \vdash M = \vec{x} : (\vec{\sigma})$. □

Example 2.4.3 The famous *cyclic I problem* can be a good test case for a formalization of graph rewriting systems. Let us consider the pure cyclic sharing theory with an operator symbol $I : (\sigma) \to (\sigma)$ and a rewriting system generated by a rule

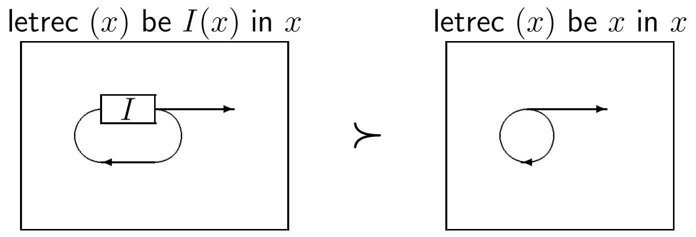
$I(x) \succ x$, which satisfies the minimality condition. Now we shall look at a cyclic term $C \equiv \mathsf{letrec}\ (x)\ \mathsf{be}\ I(x)\ \mathsf{in}\ x$. How can we rewrite this C? This is not a trivial question as it first looks like. An early attempt (Barendregt et al. [BvEG+87]) was close to the traditional term rewriting. Following the infinite unwinding semantics, one may regard this as a representation of the infinite application of I, i.e. $C = I(C) = I(I(C)) = \ldots = I(I(I\ldots))$. Having this "unwinding" $C = I(C)$ in mind,

$$C \;=\; \underline{I(C)} \;\succ\; C$$

therefore rewrite C to itself. However it was pointed out that this solution causes non-confluency (consider $\mathsf{letrec}\ (x)\ \mathsf{be}\ F(G(x))\ \mathsf{in}\ x$ with rules $F(x) \succ x$ and $G(x) \succ x$), see e.g. [AK96]. To get rid of this problem, a new solution was proposed in [AK96], respecting the circularly shared structure of C:

$$C \;\equiv\; \mathsf{letrec}\ (x)\ \mathsf{be}\ \underline{I(x)}\ \mathsf{in}\ x \;\succ\; \mathsf{letrec}\ (x)\ \mathsf{be}\ x\ \mathsf{in}\ x \;\equiv\; \bullet$$

Thus we rewrite C to the blackhole constant $\bullet$. Our rewriting system is this new version. Indeed our equational theory does not equate C to $I(C)$.



□

2.5 Equational Term Graph Rewriting

We conclude this chapter by indicating the connection between the *equational term graph rewriting* of Ariola and Klop [AK96] (see also Ariola and Arvind [AA95]) and our equational treatment of sharing graphs which was largely inspired by their work.

The equational theory underlying Ariola and Klop's work is, in spirit, essentially identical to that of ours – but not quite the same; see the discussion below. Our equational theory allows nestings of systems of equations (let/letrec-blocks) whereas theirs does not, but this is a matter of presentation. We have chosen our richer version to make the structural (algebraic) nature of sharing graphs clear, and this choice can be justified by the semantic considerations throughout the rest of this thesis.

The significance of the Ariola-Klop approach lies on the clear treatment of the bisimilarity between systems of equations and some basic operations on them, including copying, substitution and flattening. However, while they treat these operations as rewriting steps on systems of equations and have shown the confluence results, the connection with real rewriting on sharing graphs is not always simple, as some of these operations do not change the corresponding sharing graphs (and some do).

Our rewriting systems are designed not on systems of equations but on equivalence classes of them, so specify rewriting on sharing graphs themselves. In this sense our choice is more abstract; the copying and substitution operations are part of our equational theory as long as they do not change the corresponding graph; and the flattening is contained in our equational theory.

We believe that, while Ariola-Klop's results are elegant, proper computation on sharing graphs must be represented by the rewriting systems on them, and our "up to equvalence classes" (which does mean "up to sharing graphs") approach is suitable for this purpose. Also, for discussing the semantic contents of rewriting systems, it is important to separate the rewriting on representations (systems of equations) from that on the real semantic objects (sharing graphs) strictly. We hope that our direction will be more justified by further research.

Chapter 3

Models of Acyclic Sharing Theory

This chapter presents the most basic part of our development of the semantic models of sharing graphs.

Here we deal with just the acyclic case. The reader may be disappointed, as technically and practically (and conceptually!) cyclic sharing graphs are far more interesting, and also the difference between acyclic and cyclic settings are relatively small if we just look at the definition of sharing graphs - in fact we define acyclic graphs as special instances of cyclic ones, so one may wonder why we do not start our semantic story from the models of cyclic sharing.

However, it turns out that the models of acyclic cases admit a fairly simple category-theoretic formulation, which can be regarded as a generalization of the traditional treatment of algebraic theories in categorical type theory. And, as any other correspondence between syntactic theories and semantic models in mathematics and computer science, allowing stronger constructions (in this case cyclic sharing) amounts to restricting the class of models by assuming additional conditions. Starting from the models of acyclic sharing, which are fairly flexible and general, in later chapters we add new conditions to make them models of higher-order sharing graphs (the condition will be a form of *adjunction*), as well as models of cyclic sharing (the condition is a new notion called *trace*).

We believe that this systematic development is useful to identify what is needed for modeling (i.e. implementing!) sharing graphs, in a step-by-step manner. Also we think it useful to make the comparison with the traditional approach to algebraic theories as clear as we can. So, though computationally not very compelling, we describe our theory-model correspondence in the standard style of categorical type theory as found in [Cro93].

The category theory needed in this thesis is not heavy at all, but we have to use the language for monoidal categories very frequently. So we shall first summarize the material which will be used throughout the rest of this thesis. After that, we introduce our class of models, and show the expected properties such as soundness, completeness and the theory-model correspondence. Also we observe that rewriting systems on sharing theories have the semantic interpretation in our models as local preorders.

3.1 Preliminaries from Category Theory

We shall briefly recall basic definitions of monoidal categories [ML71, JS93] which will be the main technical language in the rest of this thesis.

Definition 3.1.1 (monoidal categories)
A *monoidal category* (or *tensor category*) $\mathcal{M} = (\mathcal{M}, \otimes, I, a, l, r)$ consists of a category $\mathcal{M}$, a functor $\otimes : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ (called the *monoidal* or *tensor product*), an object $I \in \mathcal{M}$ (the *unit object*) and natural isomorphisms

$$\begin{array}{lcl} a_{A,B,C} & : & (A \otimes B) \otimes C \xrightarrow{\sim} A \otimes (B \otimes C) \\ l_A & : & I \otimes A \xrightarrow{\sim} A \\ r_A & : & A \otimes I \xrightarrow{\sim} A \end{array}$$

such that, for objects $A, B, C, D \in \mathcal{M}$, the following two diagrams (called the *associativity pentagon* and the *triangle for unit* respectively) commute:

$$\begin{tikzcd}
((A \otimes B) \otimes C) \otimes D \arrow[rr, "a"] \arrow[d, "a \otimes D"'] & & (A \otimes B) \otimes (C \otimes D) \arrow[d, "a"] \\
(A \otimes (B \otimes C)) \otimes D \arrow[r, "a"'] & A \otimes ((B \otimes C) \otimes D) \arrow[r, "A \otimes a"'] & A \otimes (B \otimes (C \otimes D))
\end{tikzcd}$$

$$\begin{tikzcd}
(A \otimes I) \otimes B \arrow[rr, "a"] \arrow[dr, "r \otimes B"'] & & A \otimes (I \otimes B) \arrow[dl, "A \otimes l"] \\
& A \otimes B &
\end{tikzcd}$$

A *strict monoidal category* is a monoidal category in which all of $a_{A,B,C}$, l_A and r_A are identity arrows (hence $A \otimes (B \otimes C)$ and $(A \otimes B) \otimes C$ are the same object, and so are $A \otimes I$, $I \otimes A$ and A). □

Definition 3.1.2 (symmetric monoidal categories)
A *symmetry* for a monoidal category is a natural transformation

$$c_{A,B} : A \otimes B \to B \otimes A$$

subject to the following two commutative diagrams (the *bilinearity* and *symmetry*):

$$\begin{tikzcd}
(A \otimes B) \otimes C \arrow[r, "a"] \arrow[d, "c \otimes C"'] & A \otimes (B \otimes C) \arrow[r, "c"] & (B \otimes C) \otimes A \arrow[d, "a"] \\
(B \otimes A) \otimes C \arrow[r, "a"'] & B \otimes (A \otimes C) \arrow[r, "B \otimes c"'] & B \otimes (C \otimes A)
\end{tikzcd}$$

$$\begin{tikzcd}
A \otimes B \arrow[d, "c"'] \arrow[dr, "A \otimes B"] & \\
B \otimes A \arrow[r, "c"'] & A \otimes B
\end{tikzcd}$$

Note that the symmetry condition implies that c is a natural isomorphism. A monoidal category equipped with a symmetry is called a *symmetric monoidal category* (*SMC*). A *strict* symmetric monoidal category is then a strict monoidal category with symmetry – note that the symmetry need not be "strict", as we do not assume that $A \otimes B = B \otimes A$. □

Example 3.1.3 Here are some examples of monoidal categories:

- The category of sets and partial functions is symmetric monoidal, where the monoidal product is given by the direct product.
- Similarly, the category of sets and binary relations is symmetric monoidal with the direct product as the monoidal product.
- Abelian groups and homomorphisms form a symmetric monoidal category, where the unit object is $\mathbb{Z}$ and the monoidal product is the tensor product of abelian groups.
- Let $\mathcal{C}$ be a category. The category $[\mathcal{C}, \mathcal{C}]$ has endofunctors on $\mathcal{C}$ as objects and natural transformations between them as arrows. The composition of functors gives a strict monoidal product on $[\mathcal{C}, \mathcal{C}]$, where the identity functor serves as the unit object. This monoidal category may not have a symmetry. □

In this thesis, we frequently use strict symmetric monoidal categories naturally arising from syntactic constructions. For ease of later developments, we shall give an axiomatization of strict symmetric monoidal categories below (the labels of axioms are purely conventional).

[axioms for strict monoidal category]

M1	$f; id = f = id; f$	M4	$f; (g; h) = (f; g); h$
M2	$f \otimes id_I = f = id_I \otimes f$	M5	$f \otimes (g \otimes h) = (f \otimes g) \otimes h$
M3	$id \otimes id = id$	M6	$(f; g) \otimes (f'; g') = (f \otimes f'); (g \otimes g')$

[axioms for symmetry]

S1	$c; (f \otimes g) = (g \otimes f); c$	S3	$c_{A,B}; c_{B,A} = id_{A \otimes B}$
S2	$(c_{A,B} \otimes id_C); (id_B \otimes c_{A,C}) = c_{A,B \otimes C}$		

Example 3.1.4 (cartesian categories)
In this thesis, by a *cartesian category* we mean a category with *chosen* binary products (binary product cones) and a *chosen* terminal object. We write $A \times B$ for the chosen binary product of objects A and B, and also write $A \overset{\pi_{A,B}}{\leftarrow} A \times B \overset{\pi'_{A,B}}{\rightarrow} B$ for the chosen product cone. Given a cone $A \overset{f}{\leftarrow} C \overset{g}{\rightarrow} B$, $\langle f, g \rangle : C \to A \times B$ is then the uniquely determined arrow such that $\langle f, g \rangle; \pi_{A,B} = f$ and $\langle f, g \rangle; \pi'_{A,B} = g$. The chosen terminal object will be denoted by 1, and the terminal map is $!_A : A \to 1$.

From this information, a cartesian category is regarded as a symmetric monoidal category whose monoidal products are binary products and unit object is the terminal object:

$$\begin{array}{lcl}
f \times g & = & \langle \pi_{A,B}; f, \pi'_{A,B}; g \rangle \quad \text{for } f : A \to A' \text{ and } g : B \to B' \\
a_{A,B,C} & = & \langle \pi_{A \times B,C}; \pi_{A,B}, \langle \pi_{A \times B,C}; \pi'_{A,B}, \pi'_{A \times B,C} \rangle \rangle \\
l_A & = & \pi'_{1,A} \\
r_A & = & \pi_{A,1} \\
c_{A,B} & = & \langle \pi'_{A,B}, \pi_{A,B} \rangle
\end{array}$$

A *strict cartesian category* is a cartesian category in which all components of a, l and r are identity arrows. □

Definition 3.1.5 (monoidal functors [EK66])
For monoidal categories $\mathcal{M} = (\mathcal{M}, \otimes, I, a, l, r)$ and $\mathcal{M}' = (\mathcal{M}', \otimes', I', a', l', r')$, a *monoidal functor* from $\mathcal{M}$ to $\mathcal{M}'$ is a tuple (F, m, m_I) where F is a functor from $\mathcal{M}$ to $\mathcal{M}'$, m is a natural transformation from $F(-) \otimes' F(=)$ to $F(- \otimes =)$ and $m_I : I' \to FI$ in $\mathcal{M}'$ which satisfy the coherence conditions below.

$$\begin{tikzcd}
(FA \otimes' FB) \otimes' FC \arrow[r, "m \otimes' FC"] \arrow[d, "a'"] & F(A \otimes B) \otimes' FC \arrow[r, "m"] & F((A \otimes B) \otimes C) \arrow[d, "F(a)"] \\
FA \otimes' (FB \otimes' FC) \arrow[r, "FA \otimes' m"'] & FA \otimes' F(B \otimes C) \arrow[r, "m"'] & F(A \otimes (B \otimes C))
\end{tikzcd}$$

$$\begin{tikzcd}
I' \otimes' FA \arrow[r, "l'"] \arrow[d, "m_I \otimes FA"'] & FA & & FA \otimes' I' \arrow[r, "r'"] \arrow[d, "FA \otimes m_I"'] & FA \\
FI \otimes' FA \arrow[r, "m"'] & F(I \otimes A) \arrow[u, "Fl"'] & & FA \otimes' FI \arrow[r, "m"'] & F(A \otimes I) \arrow[u, "Fr"']
\end{tikzcd}$$

□

Definition 3.1.6 (symmetric monoidal functor [EK66])
Let $\mathcal{M} = (\mathcal{M}, \otimes, I, a, l, r, c)$ and $\mathcal{M}' = (\mathcal{M}', \otimes', I', a', l', r', c')$ be symmetric monoidal categories. A *symmetric monoidal functor* from $\mathcal{M}$ to $\mathcal{M}'$ is a monoidal functor (F, m, m_I) which additionally satisfies the following condition:

$$\begin{tikzcd}
FA \otimes' FB \arrow[r, "c'"] \arrow[d, "m"'] & FB \otimes' FA \arrow[d, "m"] \\
F(A \otimes B) \arrow[r, "Fc"'] & F(B \otimes A)
\end{tikzcd}$$

□

Lemma 3.1.7 (composition of monoidal functors [EK66])
For (symmetric) monoidal functors $(F, m, m_I) : \mathcal{M} \to \mathcal{M}'$ and $(G, n, n_{I'}) : \mathcal{M}' \to \mathcal{M}''$, $(G, n, n_{I'}) \circ (F, m, m_I) \equiv (G \circ F, G(m) \circ n_{F,F}, G(m_I) \circ n_{I'})$ is also a (symmetric) monoidal functor from $\mathcal{M}$ to $\mathcal{M}'$. This composition is associative, and satisfies the identity law for the identity (symmetric) monoidal functor. □

Definition 3.1.8 (strong/strict monoidal functors)

- A monoidal functor (F, m, m_I) is called *strong* if m is a natural isomorphism and m_I is an isomorphism.
- A monoidal functor (F, m, m_I) is called *strict* if all components of m and m_I are identity arrows. □

Example 3.1.9 (finite product preserving functors)
Let $\mathcal{C}$ be a cartesian category and $\mathcal{D}$ a category. We say that a functor $F : \mathcal{C} \to \mathcal{D}$ *preserves finite products* if F sends the chosen (hence every) product cone for A and B in $\mathcal{C}$ to a product cone for FA and FB in $\mathcal{D}$, and also maps the terminal object of $\mathcal{C}$ to a terminal object in $\mathcal{D}$. It is then routine to see that, for cartesian categories $\mathcal{C}$ and $\mathcal{D}$, a functor $F : \mathcal{C} \to \mathcal{D}$ preserves finite products if and only if it is a strong symmetric monoidal functor (with respect to the symmetric monoidal

structure described in the last example). Similarly, we say that a functor between cartesian categories *strictly preserves finite products* if it sends the chosen product cones to the chosen product cones and the chosen terminal object to the chosen terminal object. Then a functor strictly preserves finite products if and only if it is a strict symmetric monoidal functor. Note that there are symmetric monoidal functors between cartesian categories which do not preserve finite products (e.g. the covariant powerset functor on the category of sets). □

Definition 3.1.10 (monoidal natural transformations [EK66])
Given monoidal functors (F, m, m_I), (G, n, n_I) with the same source and target monoidal categories, a *monoidal natural transformation* from (F, m, m_I) to (G, n, n_I) is a natural transformation $\phi : F \to G$ such that the following diagrams commute:

$$\begin{tikzcd}
FA \otimes' FB \arrow[r, "m"] \arrow[d, "\phi_A \otimes' \phi_B"'] & F(A \otimes B) \arrow[d, "\phi_{A\otimes B}"] \\
GA \otimes' GB \arrow[r, "n"'] & G(A \otimes B)
\end{tikzcd}
\qquad
\begin{tikzcd}
 & I' \arrow[dl, "m_I"'] \arrow[dr, "n_I"] & \\
FI \arrow[rr, "\phi_I"'] & & GI
\end{tikzcd}$$

□

In the case of strict functors, a monoidal natural transformation ϕ between them is a natural transformation satisfying $\phi_{A\otimes B} = \phi_A \otimes \phi_B$ and $\phi_I = id_I$.

3.2 Acyclic Sharing Models

The correspondence between algebraic theories and finite product preserving functors, originated by Lawvere [Law63], is now standard – see [Cro93] for a detailed account. We shall briefly recall this story, to help with our descriptions of our story for sharing theories.

Given an algebraic theory $\mathbb{T}$, one can give a model M of the theory as a sound interpretation in a cartesian category $\mathcal{C}$. It is also possible to consider the category of models in $\mathcal{C}$ and homomorphisms between them, for which we shall write $\mathbf{Mod}(\mathbb{T}, \mathcal{C})$. By the way, we can construct a cartesian category from $\mathbb{T}$ in syntactic way, which, denoted by $\mathcal{C}_\mathbb{T}$ and called the *classifying category* or the *term model*, gives an obvious complete model in it. And it gives a pleasant natural equivalence

$$\mathbf{FP}(\mathcal{C}_\mathbb{T}, \mathcal{C}) \simeq \mathbf{Mod}(\mathbb{T}, \mathcal{C})$$

where **FP** is the category of small cartesian categories and finite product preserving functors. On the other hand, with any cartesian category $\mathcal{C}$, we can associate an algebraic theory and these correspondences give an equivalence of categories of theories and models.

The departure from this standard story is a simple observation that cartesian categories are not sufficient, or too strong, for interpreting the notion of sharing. We wish to separate the meaning of $\mathsf{let}\ (x)\ \mathsf{be}\ M\ \mathsf{in}\ N$ from that of $N\{M/x\}$ if M contains some uncopiable resource (operator symbol), because in $N\{M/x\}$, M can be duplicated or discarded according to the number of x's occurring in N. But this distinction is not possible in the conventional interpretation of algebraic theories in cartesian categories, as both of them are simply interpreted as a composition of arrows corresponding to M and N. Then the natural idea is to replace cartesian categories by symmetric monoidal categories whose axiomatization allows just the linear treatment of resources. However, for modeling sharing graph, we also need to treat variables (pointer-names) in a non-linear way, so we want to keep the benefit of cartesian categories. Therefore we are led to use, instead of cartesian categories, identity-on-objects, strict symmetric monoidal functors from cartesian categories to symmetric monoidal categories. Though not necessarily true, we may intuitively consider this as a symmetric monoidal category with a sub-cartesian category which is full on objects. This amalgam of linear and non-linear settings, which we will call a *cartesian-center symmetric monoidal category*, works well as models of sharing graphs. We interpret linear resources (operator symbols) directly in the symmetric monoidal category, while non-linear variables are first interpreted in the cartesian category part and then imported into the symmetric monoidal category via the strict functor.

Definition 3.2.1 (cartesian-center symmetric monoidal categories)
A *cartesian-center symmetric monoidal category* (*cartesian-center SMC*) is a strict symmetric monoidal functor $\mathcal{F}$ from a cartesian category $\mathcal{C}$ to a symmetric monoidal category $\mathcal{S}$ which is identity on objects. It is *strict* if both $\mathcal{C}$ and $\mathcal{S}$ are strict, and *faithful* if $\mathcal{F}$ is faithful (i.e. $\mathcal{C}$ is a subcategory of $\mathcal{S}$). $\Box$

Remark 3.2.2 The word "center" is taken from work by Power and Robinson [PR96, Pow96b] on *premonoidal categories*. Informally, a premonoidal category is a monoidal category without bifunctoriality – so, for $f : A \to B$ and $g : A' \to B'$, $(f \otimes A'); (B \otimes g)$ and $(A \otimes g); (f \otimes B')$ may not agree in a premonoidal category. An arrow f is called *central* if the equation above holds for any g. Generalizing the classical notion of the center of groups and monoids, Power and Robinson define the subcategory of central maps which is necessarily a monoidal category, and this is the original definition of the center of a premonoidal category. In some interesting applications, a center contains a cartesian subcategory, the inclusion being identity on objects and strict premonoidal, for instance see Thielecke's work on models of continuations [Thi97]. In this thesis we do not need the full generality of Power and Robinson's work, but we prefer to keep this connection, as there seems some natural extension of our work to premonoidal structures with important applications; see the conclusion chapter for further discussion. □

Example 3.2.3 Any cartesian category $\mathcal{C}$ is, of course, a cartesian-center symmetric monoidal category, where the symmetric monoidal category $\mathcal{S}$ is identical to the cartesian category $\mathcal{C}$, and the functor $\mathcal{F}$ is just the identity on $\mathcal{C}$. □

Example 3.2.4 Typically, one may take $\mathcal{C}$ as the category of sets and functions. To model partial computation, $\mathcal{S}$ can be the category of sets and partial functions, whose monoidal structure is inherited from the cartesian products of $\mathcal{C}$ (which are no longer cartesian in $\mathcal{S}$). To model non-determinism, $\mathcal{S}$ can be the category of sets and binary relations, again whose monoidal structure is inherited from the cartesian products of $\mathcal{C}$. Similar examples are available by replacing $\mathcal{C}$ by a category of predomains (e.g. ω-cpo's) and $\mathcal{S}$ by a suitable category of partial maps etc. Actually many such $\mathcal{S}$ arise as the Kleisli categories of commutative monads on $\mathcal{C}$; such cases will play the central role as the semantic models of the higher-order extensions in the next chapter. □

Definition 3.2.5 (cartesian-center functors)
A *cartesian-center symmetric monoidal functor* (shortly *cartesian-center functor*) between cartesian-center SMC's $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ and $\mathcal{F}' : \mathcal{C}' \to \mathcal{S}'$ consists of a pair of functors (Φ, Ψ) where $\Phi : \mathcal{C} \to \mathcal{C}'$ is a strict finite product preserving functor and $\Psi : \mathcal{S} \to \mathcal{S}'$ is a strict symmetric monoidal functor, satisfying $\mathcal{F}' \circ \Phi = \Psi \circ \mathcal{F}$. □

Definition 3.2.6 (cartesian-center natural transformations)
Let (Φ_1, Ψ_1) and (Φ_2, Ψ_2) be cartesian-center functors from $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ to $\mathcal{F}' : \mathcal{C}' \to \mathcal{S}'$. A *cartesian-center monoidal natural transformations* (*cartesian-center natural transformation* for short) from (Φ_1, Ψ_1) to (Φ_2, Ψ_2) consists of a pair of monoidal natural transformations $\alpha : \Phi_1 \stackrel{\cdot}{\to} \Phi_2$ and $\beta : \Psi_1 \stackrel{\cdot}{\to} \Psi_2$ such that $\mathcal{F}'\alpha = \beta\mathcal{F}$. (Since $\mathcal{F}$ is identity on objects, components of β are determined by α, but we require the naturality in $\mathcal{S}'$). □

We write **CcSMC** for the 2-category of small cartesian-center symmetric monoidal categories, cartesian-center functors and cartesian-center natural transformations.

It is well known that cartesian categories satisfy a functional completeness property [LS86, HJ95]. We have a parallel result for cartesian-center SMC's as a straightforward generalization (and as a special case of the 2-categorical formulation in [HJ95]):

Lemma 3.2.7 Let $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ be a cartesian-center SMC and A be an object of $\mathcal{C}$ (hence $\mathcal{S}$). Then the Kleisli category $\mathcal{C}//A$ of a comonad $A \times (-)$ on $\mathcal{C}$ is a cartesian category; the Kleisli category $\mathcal{S}//A$ of a comonad $A \otimes (-)$ on $\mathcal{S}$ is a symmetric monoidal category; and $\mathcal{F}$ induces an identity-on-objects strict symmetric monoidal functor from $\mathcal{C}//A$ to $\mathcal{S}//A$ (for which we write $\mathcal{F}//A$). Therefore $\mathcal{F}//A : \mathcal{C}//A \to \mathcal{S}//A$ is a cartesian-center SMC. $\square$

The notation $\mathcal{C}//A$ is taken from [HJ95] where it is called "the simple slice category over A". $\mathcal{F}//A$ serves as a "polynomial category" [LS86] in the following sense.

Proposition 3.2.8 (functional completeness)
The obvious identity-on-objects cartesian-center functor $(\mathcal{I}, \mathcal{J})$ from $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ to $\mathcal{F}//A : \mathcal{C}//A \to \mathcal{S}//A$ satisfies the following universal property. For any identity-on-objects cartesian-center functor (Φ, Ψ) from $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ to $\mathcal{F}' : \mathcal{C}' \to \mathcal{S}'$ with an arrow $a : 1 \to A$ in $\mathcal{C}'$, there is a unique cartesian-center functor (Φ_a, Ψ_a) from $\mathcal{F}//A : \mathcal{C}//A \to \mathcal{S}//A$ to $\mathcal{F}' : \mathcal{C}' \to \mathcal{S}'$ such that $(\Phi_a, \Psi_a) \circ (\mathcal{I}, \mathcal{J}) = (\Phi, \Psi)$ and $\Phi_a(\pi_{A.1}) = a$. $\square$

(In general the universality should be stated up to isomorphism, as done in [HJ95], but here we deal with functors which preserve structure on the nose.)

Though these observations are obvious in this basic setting, they will remain true after introducing additional requirements for interpreting higher-order and cyclic sharing, and turn out to be useful for simplifying our calculation (especially in the cyclic setting, see Lemma 6.2.3 and the proofs of Theorem 7.1.1 and Theorem 7.2.1).

Now let us consider how to interpret sharing theories in our categorical structure. We proceed as follows: define the notion of structure for a signature; give the notion of models; and then show a soundness theorem. A completeness theorem follows after constructing the classifying category in the next section.

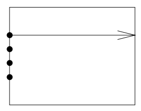
Definition 3.2.9 (sharing structures)
A *sharing structure* over an S-sorted signature Σ in a strict cartesian-center SMC

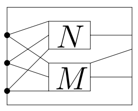
$\mathcal{F} : \mathcal{C} \to \mathcal{S}$ is a pair of functions $[\![-]\!]_S : S \to \mathrm{Obj}(\mathcal{S})$ and $[\![-]\!]_\Sigma : \Sigma \to \mathrm{Arr}(\mathcal{S})$ (subscripts may be omitted, and we just overload $[\![-]\!]$ for both of them) such that $[\![F]\!] : [\![(\sigma_1, \ldots, \sigma_m)]\!] \to [\![(\tau_1, \ldots, \tau_n)]\!]$ for each operator symbol $F : (\sigma_1, \ldots, \sigma_m) \to (\tau_1, \ldots, \tau_n)$ of Σ, where $[\![(\sigma_1, \ldots, \sigma_m)]\!] = [\![\sigma_1]\!] \otimes \ldots \otimes [\![\sigma_m]\!]$. $\square$

Given a sharing structure $[\![-]\!]$ in a strict cartesian-center SMC $F : \mathcal{C} \to \mathcal{S}$, we define $[\![x_1 : \sigma_1, \ldots, x_m : \sigma_m \vdash M : (\tau_1, \ldots, \tau_n)]\!] : [\![(\sigma_1, \ldots, \sigma_m)]\!] \to [\![(\tau_1, \ldots, \tau_n)]\!]$ in $\mathcal{S}$ for each well-typed term $x_1 : \sigma_1, \ldots, x_m : \sigma_m \vdash M : (\tau_1, \ldots, \tau_n)$ as follows, by induction on the typing rules.

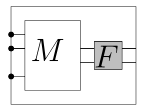
$$\begin{array}{lcl}
[\![\Gamma, x : \sigma \vdash x : (\sigma)]\!] & = & \mathcal{F}(\pi') \\
[\![\Gamma \vdash F(M) : (\vec{\tau})]\!] & = & [\![\Gamma \vdash M : (\vec{\sigma})]\!]; [\![F]\!] \\
[\![\Gamma \vdash 0 : ()]\!] & = & \mathcal{F}(!) \\
[\![\Gamma \vdash M \otimes N : (\vec{\sigma}, \vec{\tau})]\!] & = & \mathcal{F}(\Delta); ([\![\Gamma \vdash M : (\vec{\sigma})]\!] \otimes [\![\Gamma \vdash N : (\vec{\tau})]\!]) \\
[\![\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]\!] & = & \mathcal{F}(\Delta); (id \otimes [\![\Gamma \vdash M : (\vec{\sigma})]\!]); [\![\Gamma, \vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})]\!] \\
[\![\Gamma, x' : \sigma', x : \sigma, \Gamma' \vdash M : (\vec{\tau})]\!] & = & (id \otimes \mathcal{F}(c) \otimes id); [\![\Gamma, x : \sigma, x' : \sigma', \Gamma' \vdash M : (\vec{\tau})]\!]
\end{array}$$

where $\Delta_X = \langle id_X, id_X \rangle : X \to X \times X$ in $\mathcal{C}$. Intuitively, this definition corresponds to the inductive constructions of graphs, as shown by pictures below (reproduced from Chapter 2).

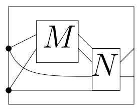

$\mathcal{F}(\pi')$



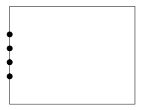
$\mathcal{F}(\Delta); ([\![\Gamma \vdash M : (\vec{\sigma})]\!] \otimes [\![\Gamma \vdash N : (\vec{\tau})]\!])$

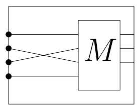


$[\![\Gamma \vdash M : (\vec{\sigma})]\!]; [\![F]\!]$



$\mathcal{F}(\Delta); (id \otimes [\![\Gamma \vdash M : (\vec{\sigma})]\!]); [\![\Gamma, \vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})]\!]$


$\mathcal{F}(!)$



$(id \otimes \mathcal{F}(c) \otimes id); [\![\Gamma, x : \sigma, x' : \sigma', \Gamma' \vdash M : (\vec{\tau})]\!]$

Note that the derivations of variables and let-bindings amount to identity arrows and composition in the simple slice category $\mathcal{S}//[\![|\Gamma|]\!]$, where $|\Gamma|$ is defined by $|x_1 : \sigma_1, \ldots, x_n : \sigma_n| = (\sigma_1, \ldots, \sigma_n)$.

Remark 3.2.10 Though a term may have many derivations, it is straightforward to show that they all give equal interpretations, and therefore the slightly loose definition above does not cause problems. $\square$

Lemma 3.2.11 Let $[\![-]\!]$ be a sharing structure. Then $[\![\vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})]\!] = [\![\vec{y} : \vec{\sigma} \vdash M\{\vec{y}/\vec{x}\} : (\vec{\tau})]\!]$ where the x's and y's are disjoint.

Proof: Induction along the construction of M. □

Definition 3.2.12 (sharing models)
A *sharing model* of an acyclic sharing theory is a sharing structure $[\![-]\!]$ in a cartesian-center SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ such that $[\![\Gamma \vdash M : (\vec{\sigma})]\!] = [\![\Gamma \vdash N : (\vec{\sigma})]\!]$ for each additional axiom $\Gamma \vdash M = N : (\vec{\sigma})$ of the theory. □

Therefore a structure is automatically a model of the pure sharing theory (which does not have any additional axiom).

Theorem 3.2.13 (soundness)
Let $[\![-]\!]$ be a sharing model of a sharing theory. If $\Gamma \vdash M = N : (\vec{\sigma})$ is derivable in the theory, then $[\![\Gamma \vdash M : (\vec{\sigma})]\!] = [\![\Gamma \vdash N : (\vec{\sigma})]\!]$.

Proof: We check the axioms in Definition 2.2.3 (page 27).

- (σ_{var}) is proved by induction on the construction of M. If M is x, this is an instance of (id). If M is either another variable or 0, it is easy to see that $[\![\Gamma \vdash \mathsf{let}\ (x)\ \mathsf{be}\ y\ \mathsf{in}\ M : (\vec{\sigma})]\!] = [\![\Gamma \vdash M : (\vec{\sigma})]\!]$. For other cases, operators:

$$\begin{array}{rll} & [\![\Gamma \vdash \mathsf{let}\ (x)\ \mathsf{be}\ y\ \mathsf{in}\ F(M) : (\vec{\tau})]\!] & \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash y : (\sigma)]\!]); [\![\Gamma, x : \sigma \vdash M]\!]; [\![F]\!] & \\ = & [\![\Gamma \vdash \mathsf{let}\ (x)\ \mathsf{be}\ y\ \mathsf{in}\ M : (\vec{\sigma})]\!]; [\![F]\!] & \\ = & [\![\Gamma \vdash M\{y/x\} : (\vec{\sigma})]\!]; [\![F]\!] & \text{ind.hyp.} \\ = & [\![\Gamma \vdash F(M)\{y/x\} : (\vec{\tau})]\!] & \end{array}$$

tensor products:

$$\begin{array}{rll} & [\![\Gamma \vdash \mathsf{let}\ (x)\ \mathsf{be}\ y\ \mathsf{in}\ M \otimes N : (\vec{\tau}, \vec{\tau}')]\!] & \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash y : (\sigma)]\!]); \mathcal{F}\Delta; & \\ & \quad ([\![\Gamma, x : \sigma \vdash M : (\vec{\tau})]\!] \otimes [\![\Gamma, x : \sigma \vdash N : (\vec{\tau}')]\!]) & \\ = & \mathcal{F}\Delta; (\mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash y : (\sigma)]\!]); [\![\Gamma, x : \sigma \vdash M : (\vec{\tau})]\!] \otimes & \\ & \quad \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash y : (\sigma)]\!]); [\![\Gamma, x : \sigma \vdash N : (\vec{\tau}')]\!]) & \\ = & \mathcal{F}\Delta; ([\![\Gamma \vdash \mathsf{let}\ (x)\ \mathsf{be}\ y\ \mathsf{in}\ M : (\vec{\tau})]\!] \otimes [\![\Gamma \vdash \mathsf{let}\ (x)\ \mathsf{be}\ y\ \mathsf{in}\ N : (\vec{\tau}')]\!]) & \\ = & \mathcal{F}\Delta; ([\![\Gamma \vdash M\{y/x\} : (\vec{\tau})]\!] \otimes [\![\Gamma \vdash N\{y/x\} : (\vec{\tau}')]\!]) & \text{ind.hyp.} \\ = & [\![\Gamma \vdash (M \otimes N)\{y/x\} : (\vec{\tau}, \vec{\tau}')]\!] & \end{array}$$

let-bindings:

$$\begin{array}{rll} & [\![\Gamma \vdash \mathsf{let}\ (x)\ \mathsf{be}\ y\ \mathsf{in}\ \mathsf{let}\ (\vec{z})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]\!] & \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash y : (\sigma)]\!]); \mathcal{F}\Delta; & \\ & \quad (id \otimes [\![\Gamma, x : \sigma \vdash M : (\vec{\sigma}')]\!]); [\![\Gamma, x : \sigma, \vec{z} : \vec{\sigma}' \vdash N : (\vec{\tau})]\!] & \\ = & \mathcal{F}\Delta; (id \otimes \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash y : (\sigma)]\!]); [\![\Gamma, x : \sigma \vdash M : (\vec{\sigma}')]\!]); & \\ & \quad \mathcal{F}\Delta; (id \otimes [\![\Gamma, \vec{z} : \vec{\sigma}' \vdash y : (\sigma)]\!]); [\![\Gamma, \vec{z} : \vec{\sigma}', x : \sigma \vdash N : (\vec{\tau})]\!] & \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash \mathsf{let}\ (x)\ \mathsf{be}\ y\ \mathsf{in}\ M : (\vec{\sigma}')]\!]); & \\ & \quad [\![\Gamma, \vec{z} : \vec{\sigma}' \vdash \mathsf{let}\ (x)\ \mathsf{be}\ y\ \mathsf{in}\ N : (\vec{\tau})]\!] & \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash M\{y/x\} : (\vec{\sigma}')]\!]); [\![\Gamma, \vec{z} : \vec{\sigma}' \vdash N\{y/x\} : (\vec{\tau})]\!] & \text{ind.hyp.} \\ = & [\![\Gamma \vdash \mathsf{let}\ (\vec{z})\ \mathsf{be}\ M\{y/x\}\ \mathsf{in}\ N\{y/x\} : (\vec{\tau})]\!] & \end{array}$$

- (id):

$$\begin{array}{ll} & [\![\Gamma \vdash \mathsf{let}\ \vec{x}\ \mathsf{be}\ M\ \mathsf{in}\ \vec{x} : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash M : (\vec{\tau})]\!]); [\![\Gamma, \vec{x} : \vec{\tau} \vdash \vec{x} : (\vec{\tau})]\!] \\ = & [\![\Gamma \vdash M : (\vec{\tau})]\!] \end{array}$$

- (ass$_1$):

$$\begin{array}{ll} & [\![\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ (\mathsf{let}\ (\vec{y})\ \mathsf{be}\ L\ \mathsf{in}\ M)\ \mathsf{in}\ N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash \mathsf{let}\ (\vec{y})\ \mathsf{be}\ L\ \mathsf{in}\ M : (\vec{\sigma}_1)]\!]); [\![\Gamma, \vec{x} : \vec{\sigma}_1 \vdash N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash L : (\vec{\sigma}_2)]\!]; [\![\Gamma, \vec{y} : \vec{\sigma}_2 \vdash M : (\vec{\sigma}_1)]\!])); \\ & \quad [\![\Gamma, \vec{x} : \vec{\sigma}_1 \vdash N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash L : (\vec{\sigma}_2)]\!]); \mathcal{F}\Delta; (id \otimes [\![\Gamma, \vec{y} : \vec{\sigma}_2 \vdash M : (\vec{\sigma}_1)]\!]); \\ & \quad [\![\Gamma, \vec{y} : \vec{\sigma}_2, \vec{x} : \vec{\sigma}_1 \vdash N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash L : (\vec{\sigma}_2)]\!]); [\![\Gamma, \vec{y} : \vec{\sigma}_2 \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]\!] \\ = & [\![\Gamma \vdash \mathsf{let}\ (\vec{y})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]\!] \end{array}$$

(This amounts to the associativity of compositions in $\mathcal{S}//[\![|\Gamma|]\!]$.)

- (ass$_2$):

$$\begin{array}{ll} & [\![\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash L : (\vec{\sigma_1})]\!]); [\![\Gamma, \vec{x} : \vec{\sigma}_1 \vdash \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash L : (\vec{\sigma_1})]\!]); \mathcal{F}\Delta; (id \otimes [\![\Gamma, \vec{x} : \vec{\sigma}_1 \vdash M : (\vec{\sigma}_2)]\!]); \\ & \quad [\![\Gamma, \vec{x} : \vec{\sigma}_1, \vec{y} : \vec{\sigma}_2 \vdash N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes \mathcal{F}\Delta; ([\![\Gamma \vdash L : (\vec{\sigma_1})]\!] \otimes [\![\Gamma \vdash M : (\vec{\sigma}_2)]\!])); \\ & \quad [\![\Gamma, \vec{x} : \vec{\sigma}_1, \vec{y} : \vec{\sigma}_2 \vdash N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash L \otimes M : (\vec{\sigma}_1, \vec{\sigma}_2)]\!]); [\![\Gamma, \vec{x} : \vec{\sigma}_1, \vec{y} : \vec{\sigma}_2 \vdash N : (\vec{\tau})]\!] \\ = & [\![\Gamma \vdash \mathsf{let}\ (\vec{x}, \vec{y})\ \mathsf{be}\ L \otimes M\ \mathsf{in}\ N : (\vec{\tau})]\!] \end{array}$$

- ($\otimes_1$):

$$\begin{array}{ll} & [\![\Gamma \vdash L \otimes (\mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N) : (\vec{\sigma}, \vec{\tau})]\!] \\ = & \mathcal{F}\Delta; ([\![\Gamma \vdash L : (\vec{\sigma})]\!] \otimes [\![\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]\!]) \\ = & \mathcal{F}\Delta; ([\![\Gamma \vdash L : (\vec{\sigma})]\!] \otimes \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash M : (\vec{\sigma}')]\!]); [\![\Gamma, \vec{x} : \vec{\sigma}' \vdash N : (\vec{\tau})]\!]) \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash M : (\vec{\sigma}')]\!]); \mathcal{F}\Delta; \\ & \quad ([\![\Gamma, \vec{x} : \vec{\sigma}' \vdash L : (\vec{\sigma})]\!] \otimes [\![\Gamma, \vec{x} : \vec{\sigma}' \vdash N : (\vec{\tau})]\!]) \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash M : (\vec{\sigma}')]\!]); [\![\Gamma, \vec{x} : \vec{\sigma}' \vdash L \otimes N : (\vec{\sigma}, \vec{\tau})]\!] \\ = & [\![\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ L \otimes N : (\vec{\sigma}, \vec{\tau})]\!] \end{array}$$

- (subst):

$$\begin{array}{ll} & [\![\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ F(N) : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash M : (\vec{\sigma}')]\!]); [\![\Gamma, \vec{x} : \vec{\sigma}' \vdash F(N) : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash M : (\vec{\sigma}')]\!]); [\![\Gamma, \vec{x} : \vec{\sigma}' \vdash N : (\vec{\sigma})]\!]; [\![F]\!] \\ = & [\![\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\sigma})]\!]; [\![F]\!] \\ = & [\![\Gamma \vdash F(\mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N) : (\vec{\tau})]\!] \end{array}$$

□

Definition 3.2.14 (homomorphisms between models)
Let $[\![-]\!]$, $[\![-]\!]'$ be sharing models of a sharing theory in a strict cartesian-center SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$. A *homomorphism* between $[\![-]\!]$ and $[\![-]\!]'$ is a family of arrows $h_\sigma : [\![\sigma]\!] \to [\![\sigma]\!]'$ for each sort σ, such that for each operator symbol $F : (\sigma_1, \dots, \sigma_m) \to (\tau_1, \dots, \tau_n)$ the following diagram commutes.

$$\begin{tikzcd}
[\![\sigma_1]\!] \otimes \dots \otimes [\![\sigma_m]\!] \arrow[r, "{[\![F]\!]}"] \arrow[d, "h_{\sigma_1} \otimes \dots \otimes h_{\sigma_m}"'] & [\![\tau_1]\!] \otimes \dots \otimes [\![\tau_n]\!] \arrow[d, "h_{\tau_1} \otimes \dots \otimes h_{\tau_n}"] \\
[\![\sigma_1]\!]' \otimes \dots \otimes [\![\sigma_m]\!]' \arrow[r, "{[\![F]\!]'}"'] & [\![\tau_1]\!]' \otimes \dots \otimes [\![\tau_n]\!]'
\end{tikzcd}$$

□

For a sharing theory $\mathbb{T}$ (determined by a set of axioms) over a signature Σ, we write $\mathbf{SharingMod}(\mathbb{T}, (\mathcal{F} : \mathcal{C} \to \mathcal{S}))$ for the category of $\mathbb{T}$'s sharing models in a strict cartesian-center SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ and the homomorphisms between models.

After introducing the notion of classifying category (term model) $\mathcal{F}_\mathbb{T} : \mathcal{C}_\mathbb{T} \to \mathcal{S}_\mathbb{T}$ for the theory $\mathbb{T}$, we will show that there is an equivalence of categories

$$\mathbf{CcSMC}((\mathcal{F}_\mathbb{T} : \mathcal{C}_\mathbb{T} \to \mathcal{S}_\mathbb{T}), (\mathcal{F} : \mathcal{C} \to \mathcal{S})) \simeq \mathbf{SharingMod}(\mathbb{T}, (\mathcal{F} : \mathcal{C} \to \mathcal{S})).$$

3.3 The Classifying Category

In this section we construct a sharing model from an acyclic sharing theory syntactically. This is done in a similar manner to the standard way to construct a term model from an algebraic theory, see for instance [Cro93].

Proposition 3.3.1 Given an acyclic sharing theory $\mathbb{T}$ over Σ, the following data give rise to a strict cartesian-center symmetric monoidal category $\mathcal{F}_\mathbb{T} : \mathcal{C}_\mathbb{T} \to \mathcal{S}_\mathbb{T}$.

- Objects are finite lists of sorts.
- An arrow of the cartesian category $\mathcal{C}_\mathbb{T}$ from $(\vec{\sigma})$ to $(\vec{\tau})$ is a term of the acyclic sharing theory of the form $\vec{x} : \vec{\sigma} \vdash \vec{y} : (\vec{\tau})$. (Note: we work up to renaming of free variables, i.e. we identify $\vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})$ with $\vec{y} : \vec{\sigma} \vdash M\{\vec{y}/\vec{x}\} : (\vec{\tau})$ for fresh variables $\vec{y}$.)

$$id_{(\sigma_1,\dots,\sigma_m)} \;=\; x_1 : \sigma_1, \dots, x_m : \sigma_m \vdash x_1 \otimes \dots \otimes x_m : (\sigma_1, \dots, \sigma_m)$$

$$(\Gamma \vdash \vec{y} : (\vec{\sigma})); (\vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})) \;=\; \Gamma \vdash N\{\vec{x}/\vec{y}\} : (\vec{\tau})$$

The terminal object is the empty list $()$, and the product of $(\vec{\sigma})$ and $(\vec{\tau})$ is $(\vec{\sigma}, \vec{\tau})$. Terminal maps, projections and pairings are given by

$$!_{(\vec{\sigma})} \; = \; \vec{x} : \vec{\sigma} \vdash 0 : ()$$

$$\begin{aligned} \pi_{(\vec{\sigma}),(\vec{\tau})} \; &= \; \vec{x} : \vec{\sigma}, \vec{y} : \vec{\tau} \vdash \vec{x} : (\vec{\sigma}) \\ \pi'_{(\vec{\sigma}),(\vec{\tau})} \; &= \; \vec{x} : \vec{\sigma}, \vec{y} : \vec{\tau} \vdash \vec{y} : (\vec{\tau}) \\ \langle \Gamma \vdash M : (\vec{\sigma}), \Gamma \vdash N : (\vec{\tau}) \rangle \; &= \; \Gamma \vdash M \otimes N : (\vec{\sigma}, \vec{\tau}) \end{aligned}$$

- An arrow of the symmetric monoidal category $\mathcal{S}_{\mathbb{T}}$ from $(\vec{\sigma})$ to $(\vec{\tau})$ is an equivalence class of terms of the theory $\vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})$, for which we shall write $[\vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})]$. Identities and compositions are

$$id_{(\sigma_1, \ldots, \sigma_m)} \; = \; [x_1 : \sigma_1, \ldots, x_m : \sigma_m \vdash x_1 \otimes \ldots \otimes x_m : (\sigma_1, \ldots, \sigma_m)]$$

$$[\Gamma \vdash M : (\vec{\sigma})]; [\vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})] \; = \; [\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]$$

The unit object is the empty list $()$, and the tensor product of $(\vec{\sigma})$ and $(\vec{\tau})$ is $(\vec{\sigma}, \vec{\tau})$. Tensor products of arrows are given by

$$[\Gamma \vdash M : (\vec{\sigma})] \otimes [\Gamma' \vdash N : (\vec{\tau})] \; = \; [\Gamma, \Gamma' \vdash M \otimes N : (\vec{\sigma}, \vec{\tau})]$$

where variables in Γ and in Γ' are disjoint.

- $\mathcal{F}_{\mathbb{T}}$ maps $\mathcal{C}_{\mathbb{T}}$'s arrow $\vec{x} : \vec{\sigma} \vdash y_1 \otimes \ldots \otimes y_n : (\vec{\tau})$ to $\mathcal{S}_{\mathbb{T}}$'s arrow $[\vec{x} : \vec{\sigma} \vdash y_1 \otimes \ldots \otimes y_n : (\vec{\tau})]$.

Proof: To check that $\mathcal{C}_{\mathbb{T}}$ forms a strict cartesian category is routine and easy – see also the remark below. We shall see that $\mathcal{S}_{\mathbb{T}}$ is a strict symmetric monoidal category, by checking that the axioms for strict symmetric monoidal categories (page 45) are satisfied.

- Identity laws (M1):

$$\begin{array}{lll} & [\Gamma \vdash M : (\vec{\sigma})]; [\vec{x} : \vec{\sigma} \vdash \vec{x} : (\vec{\sigma})] & \\ = & [\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{x} : (\vec{\sigma})] & \\ = & [\Gamma \vdash M : (\vec{\sigma})] & (\mathrm{id}) \end{array}$$

$$\begin{array}{lll} & [\vec{x} : \vec{\sigma} \vdash \vec{x} : (\vec{\sigma})]; [\vec{y} : \vec{\sigma} \vdash M : (\vec{\tau})] & \\ = & [\vec{x} : \vec{\sigma} \vdash \mathsf{let}\ (\vec{y})\ \mathsf{be}\ \vec{x}\ \mathsf{in}\ M : (\vec{\tau})] & \\ = & [\vec{x} : \vec{\sigma} \vdash M\{\vec{x}/\vec{y}\} : (\vec{\tau})] & (\sigma_{\mathrm{var}}) \\ = & [\vec{y} : \vec{\sigma} \vdash M : (\vec{\tau})] & \end{array}$$

- Unit law of tensor products (M2):

$$[\Gamma \vdash M : (\vec{\sigma})] \otimes [\vdash 0 : ()] \;=\; [\Gamma \vdash M \otimes 0 : (\vec{\sigma})] \;=\; [\Gamma \vdash M : (\vec{\sigma})]$$

  The other one is similar.

- tensor products of identities (M3):

$$[\vec{x} : \vec{\sigma} \vdash \vec{x} : (\vec{\sigma})] \otimes [\vec{y} : \vec{\tau} \vdash \vec{y} : (\vec{\tau})] \;=\; [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\tau} \vdash \vec{x} \otimes \vec{y} : (\vec{\sigma}, \vec{\tau})]$$

- Associativity of compositions (M4):

$$\begin{array}{cll}
 & ([\Gamma \vdash L : (\vec{\sigma})]; [\vec{x} : \vec{\sigma} \vdash M : (\vec{\sigma}')]); [\vec{y} : \vec{\sigma}' \vdash L : (\vec{\tau})] & \\
= & [\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M : (\vec{\sigma}')]; [\vec{y} : \vec{\sigma}' \vdash L : (\vec{\tau})] & \\
= & [\Gamma \vdash \mathsf{let}\ (\vec{y})\ \mathsf{be}\ (\mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M)\ \mathsf{in}\ N : (\vec{\tau})] & \\
= & [\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})] & (\mathrm{ass}_1) \\
= & [\Gamma \vdash L : (\vec{\sigma})]; [\vec{x} : \vec{\sigma} \vdash \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})] & \\
= & [\Gamma \vdash L : (\vec{\sigma})]; ([\vec{x} : \vec{\sigma} \vdash M : (\vec{\sigma}')]; [\vec{y} : \vec{\sigma}' \vdash L : (\vec{\tau})]) &
\end{array}$$

- Strict associativity of tensor products (M5):

$$\begin{array}{cl}
 & [\Gamma \vdash L : (\vec{\sigma})] \otimes ([\Gamma' \vdash M : (\vec{\sigma}')] \otimes [\Gamma'' \vdash N : (\vec{\sigma}'')]) \\
= & [\Gamma \vdash L : (\vec{\sigma})] \otimes [\Gamma', \Gamma'' \vdash M \otimes N : (\vec{\sigma}', \vec{\sigma}'')] \\
= & [\Gamma, \Gamma', \Gamma'' \vdash L \otimes M \otimes N : (\vec{\sigma}, \vec{\sigma}', \vec{\sigma}'')] \\
= & [\Gamma, \Gamma' \vdash L \otimes M : (\vec{\sigma}, \vec{\sigma}')] \otimes [\Gamma'' \vdash N : (\vec{\sigma}'')] \\
= & ([\Gamma \vdash L : (\vec{\sigma})] \otimes [\Gamma' \vdash M : (\vec{\sigma}')]) \otimes [\Gamma'' \vdash N : (\vec{\sigma}'')]
\end{array}$$

- Interchange law (M6):

$$\begin{array}{cll}
 & ([\Gamma \vdash M : (\vec{\sigma})]; [\vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})]) \otimes ([\Gamma' \vdash M' : (\vec{\sigma}')]; [\vec{x}' : \vec{\sigma}' \vdash N' : (\vec{\tau}')]) & \\
= & [\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})] \otimes [\Gamma' \vdash \mathsf{let}\ (\vec{x}')\ \mathsf{be}\ M'\ \mathsf{in}\ N' : (\vec{\tau}')] & \\
= & [\Gamma, \Gamma' \vdash (\mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N) \otimes (\mathsf{let}\ (\vec{x}')\ \mathsf{be}\ M'\ \mathsf{in}\ N') : (\vec{\tau}, \vec{\tau}')] & \\
= & [\Gamma, \Gamma' \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ (N \otimes (\mathsf{let}\ (\vec{x}')\ \mathsf{be}\ M'\ \mathsf{in}\ N')) : (\vec{\tau}, \vec{\tau}')] & (\otimes_2) \\
= & [\Gamma, \Gamma' \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ (\vec{x}')\ \mathsf{be}\ M'\ \mathsf{in}\ N \otimes N' : (\vec{\tau}, \vec{\tau}')] & (\otimes_1) \\
= & [\Gamma, \Gamma' \vdash \mathsf{let}\ (\vec{x}, \vec{x}')\ \mathsf{be}\ M \otimes M'\ \mathsf{in}\ N \otimes N' : (\vec{\tau}, \vec{\tau}')] & (\mathrm{ass}_2) \\
= & [\Gamma, \Gamma' \vdash M \otimes M' : (\vec{\sigma}, \vec{\sigma}')]; [\vec{x} : \vec{\sigma}, \vec{x}' : \vec{\sigma}' \vdash N \otimes N' : (\vec{\tau}, \vec{\tau}')] & \\
= & ([\Gamma \vdash M : (\vec{\sigma})] \otimes [\Gamma' \vdash M' : (\vec{\sigma}')]); ([\vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})] \otimes [\vec{x}' : \vec{\sigma}' \vdash N' : (\vec{\tau}')]) &
\end{array}$$

- Naturality of symmetry (S1):

$$\begin{array}{cll}
 & ([\vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})] \otimes [\vec{y} : \vec{\sigma}' \vdash N : (\vec{\tau}')]); [\vec{u} : \vec{\tau}, \vec{v} : \vec{\tau}' \vdash \vec{v} \otimes \vec{u} : (\vec{\tau}', \vec{\tau})] & \\
= & [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash M \otimes N : (\vec{\tau}, \vec{\tau}')]; [\vec{u} : \vec{\tau}, \vec{v} : \vec{\tau}' \vdash \vec{v} \otimes \vec{u} : (\vec{\tau}', \vec{\tau})] & \\
= & [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash \mathsf{let}\ (\vec{u}, \vec{v})\ \mathsf{be}\ M \otimes N\ \mathsf{in}\ \vec{v} \otimes \vec{u} : (\vec{\tau}', \vec{\tau})] & \\
= & [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash \mathsf{let}\ (\vec{u})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{let}\ (\vec{v})\ \mathsf{be}\ N\ \mathsf{in}\ \vec{v} \otimes \vec{u} : (\vec{\tau}', \vec{\tau})] & (\mathrm{ass}_2) \\
= & [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash \mathsf{let}\ (\vec{u})\ \mathsf{be}\ M\ \mathsf{in}\ ((\mathsf{let}\ (\vec{v})\ \mathsf{be}\ N\ \mathsf{in}\ \vec{v}) \otimes \vec{u}) : (\vec{\tau}', \vec{\tau})] & (\otimes_2) \\
= & [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash \mathsf{let}\ (\vec{u})\ \mathsf{be}\ M\ \mathsf{in}\ N \otimes \vec{u} : (\vec{\tau}', \vec{\tau})] & (\mathrm{id}) \\
= & [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash N \otimes (\mathsf{let}\ (\vec{u})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{u}) : (\vec{\tau}', \vec{\tau})] & (\otimes_1) \\
= & [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash N \otimes M : (\vec{\tau}', \vec{\tau})] & (\mathrm{id}) \\
= & [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash \mathsf{let}\ (\vec{z}, \vec{w})\ \mathsf{be}\ \vec{y} \otimes \vec{x}\ \mathsf{in}\ N\{\vec{z}/\vec{y}\} \otimes M\{\vec{w}/\vec{x}\} : (\vec{\tau}', \vec{\tau})] & (\sigma_{\mathrm{var}}) \\
= & [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash \vec{y} \otimes \vec{x} : (\vec{\sigma}', \vec{\sigma})]; [\vec{z} : \vec{\sigma}', \vec{w} : \vec{\sigma} \vdash N\{\vec{z}/\vec{y}\} \otimes M\{\vec{w}/\vec{x}\} : (\vec{\tau}', \vec{\tau})] & \\
= & [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash \vec{y} \otimes \vec{x} : (\vec{\sigma}', \vec{\sigma})]; [\vec{y} : \vec{\sigma}', \vec{x} : \vec{\sigma} \vdash N \otimes M : (\vec{\tau}', \vec{\tau})] & \\
= & [\vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash \vec{y} \otimes \vec{x} : (\vec{\sigma}', \vec{\sigma})]; ([\vec{y} : \vec{\sigma}' \vdash N : (\vec{\tau}')] \otimes [\vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})]) &
\end{array}$$

S2 and S3 follow from the calculation that $\mathcal{C}_{\mathbb{T}}$ is a cartesian category, and then $\mathcal{F}_{\mathbb{T}}$ is obviously strict symmetric monoidal. □

Remark 3.3.2 $\mathcal{C}_{\mathbb{T}}$ is equivalent to the free strict cartesian category whose objects are generated from the set of sorts S. Explicitly, an arrow from $(\sigma_1, \ldots, \sigma_m)$ to $(\tau_1, \ldots, \tau_n)$ is determined by a function f from $\{1, \ldots, n\}$ to $\{1, \ldots, m\}$ such that $\tau_i = \sigma_{f(i)}$; composition is then determined by that of functions. Thus $\mathcal{C}_{\mathbb{T}}$ does not depend on the choice of theory $\mathbb{T}$. On the other hand, the symmetric monoidal category part $\mathcal{S}_{\mathbb{T}}$ is determined by the elements of Σ as well as the axioms of $\mathbb{T}$. Note that $\mathcal{F}_{\mathbb{T}}$ may not be an inclusion, for instance the axioms of the theory may enforce two different variables to be equal. □

We say a model is *complete* if $[\![\Gamma \vdash M : (\vec{\tau})]\!] = [\![\Gamma \vdash N : (\vec{\tau})]\!]$ implies $\Gamma \vdash M = N : (\vec{\tau})$ in the theory.

Theorem 3.3.3 (completeness)
Given a theory $\mathbb{T}$, there is a complete model in $\mathcal{F}_{\mathbb{T}} : \mathcal{C}_{\mathbb{T}} \to \mathcal{S}_{\mathbb{T}}$, given by $[\![\sigma]\!] = \sigma$ and $[\![F]\!] = [\vec{x} : \vec{\sigma} \vdash F(\vec{x}) : (\vec{\tau})]$ for $F : (\vec{\sigma}) \to (\vec{\tau})$. □

Note that this model is not just complete but also *full*, in that any arrow is the image of a term.

Remark 3.3.4 The reader who followed the details of the proof may notice that the axiom (subst) is not used for showing that the classifying category is a cartesian-center SMC. It is not needed even to show completeness – but needed for fullness; without (subst) the resulting category contains some "junk" morphisms (which can be ignored to show just completeness).

In Chapter 8 we will extend the definition of operators so that they can have parameters for which arbitrary substitution may not be justified, and there these junks will get a proper interpretation and explanation. □

We call $\mathcal{F}_{\mathbb{T}} : \mathcal{C}_{\mathbb{T}} \to \mathcal{S}_{\mathbb{T}}$ the *classifying category* of the acyclic sharing theory $\mathbb{T}$, and the complete model described above the *generic model* $[\![-]\!]_G$. The names are justified by the following observation.

Lemma 3.3.5 Let $[\![-]\!]$ be a model of $\mathbb{T}$ in a strict cartesian-center SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$, and consider a cartesian-center functor $(\Phi, \Psi) : (\mathcal{F} : \mathcal{C} \to \mathcal{S}) \to (\mathcal{F}' : \mathcal{C}' \to \mathcal{S}')$. We shall define a structure $[\![-]\!]'$ in $\mathcal{F}' : \mathcal{C}' \to \mathcal{S}'$ by $[\![\sigma]\!]' = \Psi([\![\sigma]\!])$ for each sort σ and $[\![F]\!]' = \Psi([\![F]\!])$ for each operator symbol F. Then $[\![-]\!]'$ is a model of $\mathbb{T}$.

Proof: It suffices to show that $[\![\Gamma \vdash M]\!]' = \Psi([\![\Gamma \vdash M]\!])$ by induction on the construction of M. □

Proposition 3.3.6 For any model $[\![-]\!]$ of $\mathbb{T}$ in a strict cartesian-center SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$, there is a unique cartesian-center functor (Φ, Ψ) from the classifying category to $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ such that $[\![-]\!] = \Psi([\![-]\!]_G)$. Pictorially:

$$\begin{tikzcd}
\mathbb{T} \arrow[r, "\forall [\![-]\!]"] \arrow[d, "{[\![-]\!]_G}"'] & \mathcal{C} \xrightarrow{\mathcal{F}} \mathcal{S} \\
\mathcal{C}_{\mathbb{T}} \xrightarrow{\mathcal{F}_{\mathbb{T}}} \mathcal{S}_{\mathbb{T}} \arrow[ur, dotted, "{\exists! (\Phi,\Psi)}"'] &
\end{tikzcd}$$

<u>Proof:</u> Since $\mathcal{C}_{\mathbb{T}}$ is the free cartesian category, $\Phi : \mathcal{C}_{\mathbb{T}} \to \mathcal{C}$ is uniquely determined. Then Ψ is required to satisfy

$$\begin{array}{lcl}
\Psi([\Gamma, x : \sigma \vdash x : (\sigma)]) & = & \mathcal{F}(\Phi(\Gamma, x : \sigma \vdash x : (\sigma))) \\
\Psi([\Gamma \vdash \mathcal{F}(M) : (\vec{\tau})]) & = & \Psi([\Gamma \vdash M : (\vec{\sigma})]); [\![F]\!] \\
\Psi([\Gamma \vdash 0 : ()]) & = & \mathcal{F}(\Phi(\Gamma \vdash 0 : ())) \\
\Psi([\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]) & = & \mathcal{F}(\Delta); (id \otimes \Psi([\Gamma \vdash M : (\vec{\sigma})])); \\
 & & \quad \Psi([\Gamma, \vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})]) \\
\Psi([\Gamma, x : \sigma, x' : \sigma', \Gamma' \vdash M : (\vec{\tau})]) & = & (id \otimes \mathcal{F}(c) \otimes id); \\
 & & \quad \Psi([\Gamma, x' : \sigma', x : \sigma, \Gamma' \vdash M : (\vec{\tau})])
\end{array}$$

Such a Ψ exists as $[\![-]\!]$ is a model, and is uniquely determined by these equations.
□

It is easily seen that, in this setting, there is a natural bijection between the cartesian-center natural transformations and the homomorphisms, and thus we have

Theorem 3.3.7

$$\mathbf{CcSMC}((\mathcal{C}_{\mathbb{T}} \xrightarrow{\mathcal{F}_{\mathbb{T}}} \mathcal{S}_{\mathbb{T}}), (\mathcal{C} \xrightarrow{\mathcal{F}} \mathcal{S})) \simeq \mathbf{SharingMod}(\mathbb{T}, (\mathcal{C} \xrightarrow{\mathcal{F}} \mathcal{S})).$$

□

Remark 3.3.8 In the pure case, there is yet another way to construct the classifying category – from sharing graphs. We omit the fairly routine construction; the resulting cartesian-center SMC is then isomorphic to the classifying category obtained as above, and the correspondence between sharing graphs and pure acyclic sharing theory gives the isomorphism of cartesian-center SMC's. This observation provides an "algebraic" justification of Theorem 2.2.16 in the last chapter; as claimed there, the correspondence is not just a bijection but actually a structure preserving isomorphism. □

3.4 Theory-Model Correspondence

This section is to complete the analogy with the standard categorical type theory. There is no technical significance, but may be of some conceptual interest for those familiar with the standard theory. A small gap between the traditional *theory-model correspondence* and ours is pointed out.

Let $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ be a small strict cartesian-center SMC. From this, we construct an acyclic sharing theory $\mathbb{T}_{\mathcal{F}}$ as follows. For simplicity, we assume that objects are freely generated from a set S. First, sorts are given by the set S. The signature of $\mathbb{T}_{\mathcal{F}}$ is then the set of $\mathcal{S}$'s arrows, i.e. for each $f : \sigma_1 \otimes \ldots \otimes \sigma_n \to \tau_1 \otimes \ldots \otimes \tau_n$ in $\mathcal{S}$ we associate an operator symbol of type $((\sigma_1, \ldots, \sigma_m), (\tau_1, \ldots, \tau_n))$. Then $\mathbb{T}_{\mathcal{F}}$ is an acyclic sharing theory over this signature, with axioms $\Gamma \vdash M = N : (\vec{\tau})$ for all $[\![\Gamma \vdash M : (\vec{\tau})]\!] = [\![\Gamma \vdash N : (\vec{\tau})]\!]$ in $\mathcal{S}$, where $[\![-]\!]$ is determined by the obvious structure in $\mathcal{F} : \mathcal{C} \to \mathcal{S}$. By definition this $[\![-]\!]$ is a model of $\mathbb{T}_{\mathcal{F}}$ in $\mathcal{F} : \mathcal{C} \to \mathcal{S}$.

In the standard theory, the classifying category of the algebraic theory obtained from a cartesian category $\mathcal{C}$ is equivalent to the original $\mathcal{C}$. In the case of sharing categories, this is not true, because the cartesian category part of a classifying category is always a free cartesian category, while not all cartesian-center SMC's are in this form. The resulting classifying category from $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ is not $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ but the composition $\mathcal{C}_0 \to \mathcal{C} \xrightarrow{\mathcal{F}} \mathcal{S}$ where $\mathcal{C}_0$ is the free cartesian category generated from S.

If we talk about this restricted class of cartesian-center SMC's, i.e. those whose domain cartesian category is free, then we will obtain the standard theory-model correspondence. The full subcategory of the restricted models share the same initial object (actually the inclusion has a right adjoint which maps a cartesian-center SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ to the composition $\mathcal{C}_0 \to \mathcal{C} \xrightarrow{\mathcal{F}} \mathcal{S}$), hence all results developed in this chapter apply equally to the restricted version. However we have chosen the larger class of models for the following reasons:

- Most of cartesian categories arising from the natural semantic models of computation (or mathematics) are of course not free. Typically we want to choose $\mathcal{C}$ as the category of sets, as well as category of predomains etc.

- In later chapters, we assume additional structure on cartesian-center SMC's to model higher-order computation. However, none of the restricted version of cartesian-center SMC's fits this new requirement.

In fact, such a restricted class has already been studied by Power, under the name of *elementary control structures* [Pow96a], in the context of the models of action calculi (where Power also considers how to deal with parameterized operator symbols in action calculi – we will review this issue in a later chapter on action calculi).

3.5 Modeling Rewriting via Local Preorders

In Chapter 2 we defined the notion of rewriting systems on sharing theories. Following [Pow96a] we give the semantic interpretation of rewriting in sharing models.

Definition 3.5.1 A locally small strict symmetric monoidal category $\mathcal{M}$ is *locally preordered* if each homset $\mathcal{M}(X, Y)$ is equipped with a preorder $\rightsquigarrow_{X,Y}$ (we write $f \rightsquigarrow g : X \to Y$ for $f \rightsquigarrow_{X,Y} g$), and they are preserved by compositions and tensor products:

$$\frac{f \rightsquigarrow f' : X \to Y \quad g \rightsquigarrow g' : Y \to Z}{f; g \rightsquigarrow f'; g' : X \to Z}$$

$$\frac{f \rightsquigarrow f' : X \to X' \quad g \rightsquigarrow g' : Y \to Y'}{f \otimes g \rightsquigarrow f' \otimes g' : X \otimes Y \to X' \otimes Y'}$$

□

Proposition 3.5.2 There is a bijection between rewriting systems on an acyclic sharing theory $\mathbb{T}$ and local preorders on $\mathcal{S}_\mathbb{T}$ of the classifying category $\mathcal{F}_\mathbb{T} : \mathcal{C}_\mathbb{T} \to \mathcal{S}_\mathbb{T}$. □

Definition 3.5.3 (minimality condition)
Let $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ be a small cartesian-center strict SMC. A local-preorder $\rightsquigarrow$ on $\mathcal{S}$ satisfies the *minimality condition* if the images of $\mathcal{F}$ are minimal, i.e. $\mathcal{F}(g) \rightsquigarrow f$ implies $f = \mathcal{F}(g)$. □

Proposition 3.5.4 There is a bijection between rewriting systems with the minimality condition on an acyclic sharing theory $\mathbb{T}$ and local preorders with the minimality condition on $\mathcal{S}_\mathbb{T}$ of the classifying category $\mathcal{F}_\mathbb{T} : \mathcal{C}_\mathbb{T} \to \mathcal{S}_\mathbb{T}$. □

Example 3.5.5 Let $\mathcal{C}$ be (the strict equivalent of) the category of sets and functions, $\mathcal{S}$ be (the strict equivalent of) the category of sets and total relations[1] and $\mathcal{F}$ be the obvious identity-on-objects inclusion functor from $\mathcal{C}$ to $\mathcal{S}$, which strictly maps the cartesian products of $\mathcal{C}$ to the symmetric monoidal products of $\mathcal{S}$ (both

[1]A relation $R \subseteq X \times Y$ is called *total* if for any $x \in X$ there is a $y \in Y$ such that xRy.

given by the direct products of sets). We give a local preorder $\rightsquigarrow$ on $\mathcal{S}$ by the inclusion of relations, i.e. $R \rightsquigarrow R'$ iff $R \supseteq R'$. Then $\rightsquigarrow$ satisfies the minimality condition – a total relation is minimal w.r.t. $\rightsquigarrow$ if and only if it is a function. We may interpret a non-deterministic programming language in this setting. For instance, the non-determinisitic primitive `zeroORone` in Chapter 1 (page 5), together with rewriting rules `zeroORone` $\succ$ `zero` and `zeroORone` $\succ$ `one`, can be interpreted as $[\![\texttt{zeroORone}]\!] = \{0,1\} \subseteq \mathbf{N}$, which satisfies $\{0,1\} \rightsquigarrow \{0\} = [\![\texttt{zero}]\!]$ and $\{0,1\} \rightsquigarrow \{1\} = [\![\texttt{one}]\!]$ as we expect. And we have (with an adequate interpretation of $+$)

$$\begin{array}{lcl} [\![\ \vdash \texttt{zeroORone} + \texttt{zeroORone} : (\mathrm{nat})\ ]\!] & = & \{0,1,2\} \\ [\![\ \vdash \mathsf{let}\ (x)\ \mathsf{be}\ \texttt{zeroORone}\ \mathsf{in}\ x + x : (\mathrm{nat})\ ]\!] & = & \{0,2\} \end{array}$$

as intuitively explained in Chapter 1. $\Box$

Chapter 4

Higher-Order Extension

In this chapter we develop a higher-order extension of acyclic sharing theories, i.e. an extension with lambda abstractions and applications. There have been considerable research on such settings, notably higher-order graph rewriting theory, also called lambda graph rewriting systems, which are led by a practical demand for efficient implementation techniques for functional programming languages.

In some sense, it is fairly routine to enrich our theory with lambda terms. As in the standard type theory, we just add term construction rules for lambda abstractions and applications:

$$\frac{\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})}{\Gamma \vdash \lambda(\vec{x}).M : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))} \qquad \frac{\Gamma \vdash M : ((\vec{\sigma}) \Rightarrow (\vec{\tau})) \quad \Gamma \vdash N : (\vec{\sigma})}{\Gamma \vdash MN : (\vec{\tau})}$$

The main trouble is to find the *right* axiomatization for them. It easily turns out that assuming either the full β axiom or η axiom is sufficient for validating any substitutions in the theory, thus we lose the notion of sharing. Actually there is no practical justification for assuming such strong axioms. In practice, substitutions of function closures (lambda abstractions) are acceptable as they can be seen as finished computation (at least with no side effects) and copying them is in general harmless.

Fortunately, this restriction on substitutions to values (variables and lambda abstractions) matches our theory of sharing very well. In fact, later we will see that adding such higher-order constructs is a conservative extension over the original first-order sharing theory.

Remark 4.0.6 There are further possible choices in designing such calculi, which however we will not take in this thesis. First, in practical implementations, all we need is the notion of *weak reduction* in which reduction under lambda abstraction and substitution into lambda abstraction are not performed. This is what real practical interpreters do, but since we are developing the syntax and semantics for (equational) reasoning about such implementation issues, we think it natural that our equational theory proves more than the interpreters perform.

Second, there is an issue of *garbage collection* (c.f. Example 2.2.10). As we have already noticed, in dealing with sharing graphs, there can be isolated resources which cannot be accessed from other part of graphs. In many implementations of functional languages, they can be eliminated safely because such junks do not perform any side effects in future (this is not the case for concurrent languages which are full of side-effect like interactions between independent resources!). So one may wish to add axioms for eliminating isolated resources. However, we do not include such axioms a priori – they can be added to strengthen the theory and to restrict the class of models for such specific purposes (e.g. the study of implementations of functional languages), but we wish to keep our approach as general as possible. For instance, we will see that our theory perfectly covers Milner's treatment of higher-order interactive systems as *higher-order action calculi* (Chapter 8); and our models cover a properly wider range of semantic models of computation than those for pure functional computation. We just choose the axioms we need, and we believe this results in a better understanding of the computation we are talking about. □

4.1 Higher-Order Acyclic Sharing Theory

Definition 4.1.1 (higher-order sorts)
Given a set of sorts S, we define the set of *higher-order sorts* S^H by

$$S^H \ni s, t, \ldots \quad ::= \quad \sigma \mid m \Rightarrow n$$

where σ ranges over S and m, n over finite lists of elements of S^H. □

Definition 4.1.2 (higher-order acyclic sharing theory)
A *higher-order acyclic sharing theory* over Σ is an equational theory on the well-typed terms with sorts S^H closed under the term construction of acyclic sharing theories plus

$$\frac{\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})}{\Gamma \vdash \lambda(\vec{x}).M : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))} \text{ abstraction}$$

$$\frac{\Gamma \vdash M : ((\vec{\sigma}) \Rightarrow (\vec{\tau})) \quad \Gamma \vdash N : (\vec{\sigma})}{\Gamma \vdash MN : (\vec{\tau})} \text{ application}$$

where the equality on terms is a congruence relation containing the axioms for the acyclic sharing theory plus the following additional axioms.

$$
\boxed{\begin{array}{lllll}
(\beta) & (\lambda(\vec{x}).M)N & = & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ N\ \mathsf{in}\ M \\
(\eta_0) & \lambda(\vec{x}).y(\vec{x}) & = & y \\
(\sigma_v) & \mathsf{let}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).M\ \mathsf{in}\ N & = & N\{\lambda(\vec{y}).M/x\} \\
(\mathrm{app}_1) & (\mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M)N & = & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ MN \\
(\mathrm{app}_2) & L(\mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N) & = & \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ LN
\end{array}}
$$

Note that the congruence must be closed under the new term constructions (lambda abstractions and applications):

$$
\frac{\Gamma \vdash M = M' : ((\vec{\sigma}) \Rightarrow (\vec{\tau})) \quad \Gamma \vdash N = N' : (\vec{\sigma})}{\Gamma \vdash MN = M'N' : (\vec{\tau})} \qquad \frac{\Gamma \vdash M = N : (\vec{\tau})}{\Gamma \vdash \lambda(\vec{x}).M = \lambda(\vec{x}).N : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))}
$$

□

Definition 4.1.3 (values)
Values are well-typed terms generated by the following grammar.

$$
V, W \ ::= \ 0 \mid x \mid \lambda(\vec{x}).M \mid V \otimes W
$$

□

The "call-by-value" $\beta\eta$ equations as well as the α-conversion of lambda-bound variables follow immediately from our axioms:

Lemma 4.1.4 The following equations are derivable in higher-order acyclic sharing theories.

$$
\begin{array}{lllll}
(\beta_v) & (\lambda(\vec{x}).M)V & = & M\{V/\vec{x}\} & \\
(\eta_v) & (\lambda(\vec{x}).V(\vec{x})) & = & V & (\vec{x} \notin FV(V)) \\
(\alpha) & \lambda(\vec{x}).M & = & \lambda(\vec{y}).M\{\vec{y}/\vec{x}\} & (\vec{y} \text{ are fresh})
\end{array}
$$

□

4.2 Higher-Order Acyclic Sharing Models

Since the higher-order acyclic sharing theories are obtained by adding new terms and axioms to the first-order ones, their models should be a subclass of those for first-order theories. We add a new requirement to cartesian-center SMC's:

Definition 4.2.1 (cartesian centrally closed SMC)
A *cartesian centrally closed symmetric monoidal category* (*cartesian centrally closed SMC*) is a cartesian-center symmetric monoidal category $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ such that, for each object X of $\mathcal{S}$, the functor $\mathcal{F}(-) \otimes X : \mathcal{C} \to \mathcal{S}$ has a right adjoint.
□

We write $X \Rightarrow (-) : \mathcal{S} \to \mathcal{C}$ for a chosen right adjoint of $\mathcal{F}(-) \otimes X$, and use $\mathbf{ap} : (X \Rightarrow A) \otimes X \to A$ in $\mathcal{S}$ for the counit of adjunction, as well as $f^* : A \to X \Rightarrow B$ in $\mathcal{C}$ for the adjunct of $f : A \otimes X \to B$ in $\mathcal{S}$. However, we often omit the information of the chosen right adjoints if there is no confusion. We note that $\mathcal{F}$ itself has a right adjoint $I \Rightarrow (-)$.

Remark 4.2.2 The word "centrally closed" is taken from the work by Power [Pow96b]. In his more general framework, we are talking about special instances of centrally closed premonoidal $\to$-categories. $\Box$

Example 4.2.3 Any cartesian closed category can be seen as a cartesian centrally closed symmetric monoidal category. $\Box$

Example 4.2.4 Let $\mathcal{C}$ be a cartesian closed category and T be a commutative monad (see Definition 5.1.3) on $\mathcal{C}$. Putting $\mathcal{S} = \mathcal{C}_T$ (the Kleisli category of T) and letting $\mathcal{F}$ be the canonical identity-on-object functor from $\mathcal{C}$ to $\mathcal{S}$, we obtain a cartesian centrally closed SMC, where the right adjoint $X \Rightarrow (-)$ is given by $(T-)^X$. The relation with this monad-based account will be further spelled out in the next chapter. $\Box$

Example 4.2.5 As an instance of the setting described above, we shall consider the case where $\mathcal{C}$ is the category of sets and functions and T is the covariant powerset functor. In this case, $\mathcal{S} = \mathcal{C}_T$ is the category of sets and binary relations (i.e. non-deterministic functions), and $X \Rightarrow Y$ is simply the set of all relations between X and Y. $\Box$

Definition 4.2.6 (cartesian centrally closed functors)
A *cartesian centrally closed functor* between cartesian centrally closed SMC's $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ and $\mathcal{F}' : \mathcal{C}' \to \mathcal{S}'$ is a cartesian-center functor (Φ, Ψ) between them such that (Φ, Ψ) is a map of adjoints [ML71] from $\mathcal{F}(-) \otimes X \dashv X \Rightarrow (-)$ to $\mathcal{F}'(-) \otimes' \Psi X \dashv \Psi X \Rightarrow' (-)$ for each object X, i.e. the following diagrams commute

(there are many equivalent formulations of this condition – see [ML71]).

$$\begin{tikzcd}
\mathcal{S} \arrow[r, "X\Rightarrow(-)"] \arrow[d, "\Psi"'] & \mathcal{C} \arrow[d, "\Phi"] \\
\mathcal{S}' \arrow[r, "\Psi X\Rightarrow'(-)"'] & \mathcal{C}'
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathcal{S}(\mathcal{F}(A)\otimes X, B) \arrow[r, "(-)^*"] \arrow[d, "\Psi"'] & \mathcal{C}(A, X\Rightarrow B) \arrow[d, "\Phi"] \\
\mathcal{S}'(\Psi(\mathcal{F}(A)\otimes X), \Psi B) \arrow[d, "||"'] & \mathcal{C}'(\Phi A, \Phi(X\Rightarrow B)) \arrow[d, "||"] \\
\mathcal{S}'(\mathcal{F}'(\Phi A)\otimes' \Psi X, \Psi B) \arrow[r, "(-)^*"'] & \mathcal{C}'(\Phi A, \Psi X\Rightarrow'\Psi B)
\end{tikzcd}$$

□

We write **CccSMC** for the 2-category of small cartesian centrally closed symmetric monoidal categories, cartesian centrally closed functors and cartesian-center natural isomorphisms.[1]

As in the last chapter (Lemma 3.2.7 and Proposition 3.2.8) we have a functional completeness result for cartesian centrally closed SMC's:

Lemma 4.2.7 Let $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ be a cartesian-center SMC and A be an object of $\mathcal{C}$ (hence $\mathcal{S}$). Then $\mathcal{F}//A : \mathcal{C}//A \to \mathcal{S}//A$ is a cartesian centrally-closed SMC. □

Proposition 4.2.8 The obvious identity-on-objects cartesian centrally closed functor $(\mathcal{I}, \mathcal{J})$ from $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ to $\mathcal{F}//A : \mathcal{C}//A \to \mathcal{S}//A$ satisfies the following universal property. For any identity-on-objects cartesian centrally closed functor (Φ, Ψ) from $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ to $\mathcal{F}' : \mathcal{C}' \to \mathcal{S}'$ with an arrow $a : 1 \to A$ in $\mathcal{C}'$, there is a unique cartesian centrally closed functor (Φ_a, Ψ_a) from $\mathcal{F}//A : \mathcal{C}//A \to \mathcal{S}//A$ to $\mathcal{F}' : \mathcal{C}' \to \mathcal{S}'$ such that $(\Phi_a, \Psi_a) \circ (\mathcal{I}, \mathcal{J}) = (\Phi, \Psi)$ and $\Phi_a(\pi_{A.1}) = a$. □

As the first-order case, we introduce the notions of structures and models, and show a soundness property.

Definition 4.2.9 (higher-order sharing structures)
A *higher-order sharing structure* over an S-sorted signature Σ is a sharing structure (Definition 3.2.9) in a strict cartesian centrally closed SMC. □

[1] Because of contravariance, for establishing the correspondence with models (Theorem 4.3.3) we need to restrict our attention to isomorphisms; see Chapter 4 of [Cro93] for a related discussion.

Given a higher-order sharing structure $[\![-]\!]$ in a strict cartesian centrally closed SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$, we define $[\![x_1 : \sigma_1, \ldots, x_m : \sigma_m \vdash M : (\tau_1, \ldots, \tau_n)]\!] : [\![\sigma_1]\!] \otimes \ldots \otimes [\![\sigma_m]\!] \to [\![\tau_1]\!] \otimes \ldots \otimes [\![\tau_n]\!]$ in $\mathcal{S}$ for each well-typed term $x_1 : \sigma_1, \ldots, x_m : \sigma_m \vdash M : (\tau_1, \ldots, \tau_n)$ as follows, by induction on the typing rules.

$$\begin{array}{lcl}
[\![\Gamma, x : \sigma \vdash x : (\sigma)]\!] & = & \mathcal{F}(\pi') \\
[\![\Gamma \vdash F(M) : (\vec{\tau})]\!] & = & [\![\Gamma \vdash M : (\vec{\sigma})]\!]; [\![F]\!] \\
[\![\Gamma \vdash 0 : ()]\!] & = & \mathcal{F}(!) \\
[\![\Gamma \vdash M \otimes N : (\vec{\sigma}, \vec{\tau})]\!] & = & \mathcal{F}(\Delta); ([\![\Gamma \vdash M : (\vec{\sigma})]\!] \otimes [\![\Gamma \vdash N : (\vec{\tau})]\!]) \\
[\![\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]\!] & = & \mathcal{F}(\Delta); (id \otimes [\![\Gamma \vdash M : (\vec{\sigma})]\!]); [\![\Gamma, \vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})]\!] \\
[\![\Gamma \vdash \lambda(\vec{x}).M : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]\!] & = & F([\![\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})]\!]^*) \\
[\![\Gamma \vdash MN : (\vec{\tau})]\!] & = & \mathcal{F}(\Delta); ([\![\Gamma \vdash M : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]\!] \otimes [\![\Gamma \vdash N : (\vec{\sigma})]\!]); \mathbf{ap} \\
[\![\Gamma, x' : \sigma', x : \sigma, \Gamma' \vdash M : (\vec{\tau})]\!] & = & (id \otimes \mathcal{F}(c) \otimes id); [\![\Gamma, x : \sigma, x' : \sigma', \Gamma' \vdash M : (\vec{\tau})]\!]
\end{array}$$

Lemma 4.2.10 Let $[\![-]\!]$ be a higher-order sharing structure. Then $[\![\vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})]\!] = [\![\vec{y} : \vec{\sigma} \vdash M\{\vec{y}/\vec{x}\} : (\vec{\tau})]\!]$ where the x's and y's are disjoint.

Proof: Induction on the construction of M. □

Definition 4.2.11 (higher-order sharing models)
A *higher-order sharing model* of a higher-order acyclic sharing theory is a higher-order sharing structure $[\![-]\!]$ in a strict cartesian centrally closed SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ such that $[\![\Gamma \vdash M : (\vec{\sigma})]\!] = [\![\Gamma \vdash N : (\vec{\sigma})]\!]$ for each additional $\Gamma \vdash M = N : (\vec{\sigma})$ in the axioms of the theory. □

Thus, again, a higher-order sharing structure automatically induces a model of the pure higher-order acyclic sharing theory.

Theorem 4.2.12 (soundness)
Let $[\![-]\!]$ be a higher-order sharing model of a higher-order sharing theory. If $\Gamma \vdash M = N : (\vec{\sigma})$ is derivable in the theory, then $[\![\Gamma \vdash M : (\vec{\sigma})]\!] = [\![\Gamma \vdash N : (\vec{\sigma})]\!]$.

Proof: We check the additional axioms given in Definition 4.1.2.

- (β):

$$\begin{array}{cl}
 & [\![\Gamma \vdash (\lambda(\vec{x}).M)N : (\vec{\tau})]\!] \\
= & \mathcal{F}\Delta; (\mathcal{F}([\![\Gamma \vdash \lambda(\vec{x}).M : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]\!]^*) \otimes [\![\Gamma \vdash N : (\vec{\sigma})]\!]); \mathbf{ap} \\
= & \mathcal{F}\Delta; (\mathcal{F}([\![\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})]\!]^*) \otimes [\![\Gamma \vdash N : (\vec{\sigma})]\!]); \mathbf{ap} \\
= & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash N : (\vec{\sigma})]\!]); [\![\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})]\!] \\
= & [\![\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ N\ \mathsf{in}\ M : (\vec{\tau})]\!]
\end{array}$$

- (η_0):

$$\begin{array}{rl} & [\![\Gamma \vdash \lambda(\vec{x}).y\vec{x} : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]\!] \\ = & \mathcal{F}([\![\Gamma, \vec{x} : \vec{\sigma} \vdash y\vec{x}(\vec{\tau})]\!]^*) \\ = & \mathcal{F}((\mathcal{F}\Delta; ([\![\Gamma, \vec{x} : \vec{\sigma} \vdash y : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]\!] \otimes [\![\Gamma, \vec{x} : \vec{\sigma} \vdash \vec{x} : (\vec{\sigma})]\!]); \mathbf{ap})^*) \\ = & \mathcal{F}((([\![\Gamma \vdash y : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]\!] \otimes [\![\vec{x} : \vec{\sigma} \vdash \vec{x} : (\vec{\sigma})]\!]); \mathbf{ap})^*) \\ = & [\![\Gamma \vdash y : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]\!]; \mathcal{F}(\mathbf{ap}^*) \\ = & [\![\Gamma \vdash y : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]\!] \end{array}$$

- (σ_v):

$$\begin{array}{rl} & [\![\Gamma \vdash \mathsf{let}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).M\ \mathsf{in}\ N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash \lambda(\vec{y}).M : ((\vec{\sigma}) \Rightarrow (\vec{\sigma}'))]\!]); [\![\Gamma, x : (\vec{\sigma}) \Rightarrow (\vec{\sigma}') \vdash N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; (id \otimes \mathcal{F}([\![\Gamma, \vec{y} : \vec{\sigma} \vdash M : (\vec{\sigma}')]\!]^*)); [\![\Gamma, x : (\vec{\sigma}) \Rightarrow (\vec{\sigma}') \vdash N : (\vec{\tau})]\!] \end{array}$$

  The rest is similar to the case of (σ_{var}), by induction on the construction of N.

- (app_1):

$$\begin{array}{rl} & [\![\Gamma \vdash (\mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M)N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta; ([\![\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]\!] \otimes [\![\Gamma \vdash N : (\vec{\sigma})]\!]); \mathbf{ap} \\ = & \mathcal{F}\Delta; ((\mathcal{F}\Delta; (id \otimes \Gamma \vdash L : (\vec{\sigma}')); [\![\Gamma, \vec{x} : \vec{\sigma}' \vdash M : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]\!]) \\ & \qquad \otimes [\![\Gamma \vdash N : (\vec{\sigma})]\!]); \mathbf{ap} \\ = & \mathcal{F}\Delta; (id \otimes \Gamma \vdash L : (\vec{\sigma}')); \mathcal{F}\Delta; \\ & \qquad ([\![\Gamma, \vec{x} : \vec{\sigma}' \vdash M : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]\!] \otimes [\![\Gamma, \vec{x} : \vec{\sigma}' \vdash N : (\vec{\sigma})]\!]); \mathbf{ap} \\ = & \mathcal{F}\Delta; (id \otimes \Gamma \vdash L : (\vec{\sigma}')); [\![\Gamma, \vec{x} : \vec{\sigma}' \vdash MN : (\vec{\tau})]\!] \\ = & [\![\Gamma \vdash \mathsf{let}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ MN : (\vec{\tau})]\!] \end{array}$$

- (app_2) is similar to (app_1).

□

For a higher-order sharing theory $\mathbb{T}$ (determined by a set of axioms) over a signature Σ, we write $\mathbf{HSharingMod}(\mathbb{T}, (\mathcal{F} : \mathcal{C} \to \mathcal{S}))$ for the category of $\mathbb{T}$'s higher-order sharing models in a strict cartesian centrally closed SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ and the homomorphisms whose components are isomorphisms (see the footnote in page 66).

4.3 The Classifying Category

As in the first-order case, we construct a term model from a higher-order sharing theory.

Proposition 4.3.1 Given a higher-order acyclic sharing theory $\mathbb{T}$, we construct a strict cartesian centrally closed SMC $\mathcal{F}_{\mathbb{T}} : \mathcal{C}_{\mathbb{T}} \to \mathcal{S}_{\mathbb{T}}$ as follows. $\mathcal{S}_{\mathbb{T}}$ is given in the same way as the first-order case, thus arrows are equivalence classes of well-typed terms with contexts. $\mathcal{C}_{\mathbb{T}}$ is the subcategory of $\mathcal{S}_{\mathbb{T}}$ whose arrows are equivalence classes of well-typed values (Definition 4.1.3) with contexts. $\mathcal{F}_{\mathbb{T}}$ is then the inclusion functor. $\square$

Proof: To see that $\mathcal{C}_{\mathbb{T}}$ is a (strict) cartesian category is straightforward. The essential point is the verification of centrally closedness. For each object $(\vec{\sigma})$ define $(\vec{\sigma})\Rightarrow(-) : \mathcal{S}_{\mathbb{T}} \to \mathcal{C}_{\mathbb{T}}$ by

$$(\vec{\sigma})\Rightarrow[\vec{x} : \vec{\sigma}' \vdash M : (\vec{\tau})] \;=\; [f : (\vec{\sigma})\Rightarrow(\vec{\sigma}') \vdash \lambda(\vec{y}).(\mathsf{let}\ (\vec{x})\ \mathsf{be}\ f(\vec{y})\ \mathsf{in}\ M) : ((\vec{\sigma})\Rightarrow(\vec{\tau}))].$$

We need to check that $(\vec{\sigma})\Rightarrow(-)$ is indeed a functor:

$$\begin{array}{lll}
& (\vec{\sigma})\Rightarrow[\vec{x} : \vec{\tau} \vdash \vec{x} : (\vec{\tau})] & \\
= & [f : (\vec{\sigma})\Rightarrow(\vec{\tau}) \vdash \lambda(\vec{y}).(\mathsf{let}\ (\vec{x})\ \mathsf{be}\ f(\vec{y})\ \mathsf{in}\ \vec{x}) : ((\vec{\sigma})\Rightarrow(\vec{\tau}))] & \\
= & [f : (\vec{\sigma})\Rightarrow(\vec{\tau}) \vdash \lambda(\vec{y}).f(\vec{y}) : ((\vec{\sigma})\Rightarrow(\vec{\tau}))] & (\mathrm{id}) \\
= & [f : (\vec{\sigma})\Rightarrow(\vec{\tau}) \vdash f : ((\vec{\sigma})\Rightarrow(\vec{\tau}))] & (\eta_0)
\end{array}$$

$$\begin{array}{lll}
& ((\vec{\sigma})\Rightarrow[\vec{x} : \vec{\tau} \vdash M : (\vec{\tau}')]);((\vec{\sigma})\Rightarrow[\vec{y} : \vec{\tau}' \vdash N : (\vec{\tau}'')]) & \\
= & [f : (\vec{\sigma})\Rightarrow(\vec{\tau}) \vdash \lambda(\vec{u}).(\mathsf{let}\ (\vec{x})\ \mathsf{be}\ f(\vec{u})\ \mathsf{in}\ M) : ((\vec{\sigma})\Rightarrow(\vec{\tau}'))]; & \\
& \quad [g : (\vec{\sigma})\Rightarrow(\vec{\tau}') \vdash \lambda(\vec{v}).(\mathsf{let}\ (\vec{y})\ \mathsf{be}\ g(\vec{v})\ \mathsf{in}\ N) : ((\vec{\sigma})\Rightarrow(\vec{\tau}''))] & \\
= & [f : (\vec{\sigma})\Rightarrow(\vec{\tau}) \vdash & \\
& \quad \mathsf{let}\ g\ \mathsf{be}\ \lambda(\vec{u}).(\mathsf{let}\ (\vec{x})\ \mathsf{be}\ f(\vec{u})\ \mathsf{in}\ M)\ \mathsf{in}\ \lambda(\vec{v}).(\mathsf{let}\ (\vec{y})\ \mathsf{be}\ g(\vec{v})\ \mathsf{in}\ N) & \\
& \qquad : ((\vec{\sigma})\Rightarrow(\vec{\tau}''))] & \\
= & [f : (\vec{\sigma})\Rightarrow(\vec{\tau}) \vdash & \\
& \quad \lambda(\vec{v}).(\mathsf{let}\ (\vec{y})\ \mathsf{be}\ (\lambda(\vec{u}).(\mathsf{let}\ (\vec{x})\ \mathsf{be}\ f(\vec{u})\ \mathsf{in}\ M))(\vec{v})\ \mathsf{in}\ N) & \\
& \qquad : ((\vec{\sigma})\Rightarrow(\vec{\tau}''))] & (\sigma_v) \\
= & [f : (\vec{\sigma})\Rightarrow(\vec{\tau}) \vdash & \\
& \quad \lambda(\vec{v}).(\mathsf{let}\ (\vec{y})\ \mathsf{be}\ (\mathsf{let}\ (\vec{x})\ \mathsf{be}\ f(\vec{v})\ \mathsf{in}\ M)\ \mathsf{in}\ N) : ((\vec{\sigma})\Rightarrow(\vec{\tau}''))] & (\beta_v) \\
= & [f : (\vec{\sigma})\Rightarrow(\vec{\tau}) \vdash & \\
& \quad \lambda(\vec{v}).(\mathsf{let}\ (\vec{x})\ \mathsf{be}\ f(\vec{v})\ \mathsf{in}\ \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N) : ((\vec{\sigma})\Rightarrow(\vec{\tau}''))] & (\mathrm{ass}_1) \\
= & (\vec{\sigma})\Rightarrow[\vec{x} : \vec{\sigma} \vdash \mathsf{let}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau}'')] & \\
= & (\vec{\sigma})\Rightarrow([\vec{x} : \vec{\tau} \vdash M : (\vec{\tau}')];[\vec{y} : \vec{\tau}' \vdash N : (\vec{\tau}'')]) &
\end{array}$$

Now we check that $(\vec{\sigma})\Rightarrow(-)$ is a right adjoint of $\mathcal{F}_{\mathbb{T}}(-)\otimes(\vec{\sigma})$. Given $f = [\Gamma, \vec{x} : \vec{\sigma} \vdash M : \vec{\tau}]$, define $f^* = [\Gamma \vdash \lambda(\vec{x}).M : ((\vec{\sigma})\Rightarrow(\vec{\tau}))]$. Also define $\mathbf{ap} = [\![f : ((\vec{\sigma})\Rightarrow(\vec{\tau})), \vec{x} : \vec{\sigma} \vdash f(\vec{x}) : (\vec{\tau})]\!]$. Then $(\mathcal{F}_{\mathbb{T}}(f^*)\otimes id);\mathbf{ap} = f$ as

$$\begin{array}{lll}
& LHS & \\
= & [\Gamma, \vec{y} : \vec{\sigma} \vdash \mathsf{let}\ (f, \vec{z})\ \mathsf{be}\ (\lambda(\vec{x}).M)\otimes\vec{y}\ \mathsf{in}\ f(\vec{z}) : (\vec{\tau})] & \\
= & [\Gamma, \vec{y} : \vec{\sigma} \vdash \mathsf{let}\ (f)\ \mathsf{be}\ \lambda(\vec{x}).M\ \mathsf{in}\ \mathsf{let}\ (\vec{z})\ \mathsf{be}\ \vec{y}\ \mathsf{in}\ f(\vec{z}) : (\vec{\tau})] & \\
= & [\Gamma, \vec{y} : \vec{\sigma} \vdash (\lambda(\vec{x}).M)(\vec{y}) : (\vec{\tau})] & (\sigma_{\mathrm{var}}), (\sigma_v) \\
= & [\Gamma, \vec{y} : \vec{\sigma} \vdash M\{\vec{y}/\vec{x}\} : (\vec{\tau})] & (\beta_v) \\
= & RHS &
\end{array}$$

Also, for $g = [\Gamma \vdash V : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))]$ in $\mathcal{C}_{\mathbb{T}}$ (thus V is equal to either a variable or a lambda abstraction), we show that $((\mathcal{F}_{\mathbb{T}}(g) \otimes id_{(\vec{\sigma})}); \mathbf{ap})^* = g$, which completes the proof of adjointness.

$$\begin{array}{lll} & LHS & \\ = & [\Gamma, \vec{x} : \vec{\sigma} \vdash \mathsf{let}\ (f, \vec{y})\ \mathsf{be}\ V \otimes \vec{x}\ \mathsf{in}\ f(\vec{y}) : (\vec{\tau})]^* & \\ = & [\Gamma, \vec{x} : \vec{\sigma} \vdash V(\vec{x}) : (\vec{\tau})]^* & (\mathrm{ass}_2), (\sigma_v) \text{ and } (\sigma_{\mathrm{var}}) \\ = & [\Gamma \vdash \lambda(\vec{x}).V(\vec{x}) : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))] & \\ = & RHS & (\eta_v) \end{array}$$

□

Now we have parallel results to the first-order case:

Theorem 4.3.2 (completeness)
Given a higher-order acyclic sharing theory $\mathbb{T}$, there is a complete model in $\mathcal{F}_{\mathbb{T}} : \mathcal{C}_{\mathbb{T}} \to \mathcal{S}_{\mathbb{T}}$, given by $[\![\sigma]\!] = \sigma$ and $[\![F]\!] = [\vec{x} : \vec{\sigma} \vdash F(\vec{x}) : (\vec{\tau})]$ for $F : (\vec{\sigma}) \to (\vec{\tau})$. □

Theorem 4.3.3

$$\mathbf{CccSMC}((\mathcal{C}_{\mathbb{T}} \xrightarrow{\mathcal{F}_{\mathbb{T}}} \mathcal{S}_{\mathbb{T}}), (\mathcal{C} \xrightarrow{\mathcal{F}} \mathcal{S})) \simeq \mathbf{HSharingMod}(\mathbb{T}, (\mathcal{C} \xrightarrow{\mathcal{F}} \mathcal{S})).$$

□

Chapter 5

Relating Models

One advantage of our axiomatic approach to the semantic models of sharing theories is that it enables us to compare similar systems arising from computer science by relating the classes of models. In this chapter we give a few case studies: we relate the first-order acyclic sharing theories and the higher-order ones; and our sharing theories with those of Moggi's *notions of computation* [Mog88, Mog91]; and also with *intuitionistic linear type theory* [Bar96, Ben95, Bie95].

All the techniques involved here are fairly standard (though we review all the needed notions below), and we believe that our semantic comparisons give a clear account of often too complicated syntactic translations and considerations.

The content of this chapter is relatively independent of the later chapters. The semantic proofs of the conservativity results are also reported in the joint work with Barber, Gardner and Plotkin [BGHP96, BGHP97, GH97] for the corresponding action calculi (see Chapter 8). The detailed descriptions of syntactic translations are found in these papers.

5.1 Preliminaries from Category Theory

We supply additional material from category theory which is needed in this chapter.

Definition 5.1.1 (monoidal adjunction)
A *(symmetric) monoidal adjunction* between (symmetric) monoidal categories is an adjunction in which both of the functors are (symmetric) monoidal and the unit and counit are monoidal natural transformations. □

Proposition 5.1.2 (Kelly [Kel74])
The left adjoint part of a (symmetric) monoidal adjunction is necessarily strong

(symmetric) monoidal. Conversely, if a strong (symmetric) monoidal functor has a right adjoint, then the adjunction is (symmetric) monoidal. □

Definition 5.1.3 (strong monad [Koc70])
A *strong monad* over a symmetric monoidal category $\mathcal{M} = (\mathcal{M}, \otimes, I, a, l, r, c)$ is a monad (T, η, μ) on $\mathcal{M}$ with a natural transformation (called a *tensorial strength*)

$$\theta_{A,X} : A \otimes TX \to T(A \otimes X)$$

subject to the following commutative diagrams.

$$\begin{tikzcd}
I \otimes TX \arrow[r, "\theta_{I,X}"] \arrow[dr, "l"'] & T(I \otimes X) \arrow[d, "Tl"] \\
& TX
\end{tikzcd}$$

$$\begin{tikzcd}
(A \otimes B) \otimes TX \arrow[rr, "\theta_{A\otimes B,X}"] \arrow[d, "a"'] & & T((A \otimes B) \otimes X) \arrow[d, "Ta"] \\
A \otimes (B \otimes TX) \arrow[r, "A\otimes\theta_{B,X}"'] & A \otimes T(B \otimes X) \arrow[r, "\theta_{A,B\otimes X}"'] & T(A \otimes (B \otimes X))
\end{tikzcd}$$

$$\begin{tikzcd}
A \otimes X \arrow[d, "A\otimes\eta_X"'] \arrow[dr, "\eta_{A\otimes X}"] & \\
A \otimes TX \arrow[r, "\theta_{A,X}"'] & T(A \otimes X)
\end{tikzcd}
\qquad
\begin{tikzcd}
A \otimes T^2X \arrow[r, "\theta_{A,TX}"] \arrow[d, "A\otimes\mu_X"'] & T(A \otimes TX) \arrow[r, "T\theta_{A,X}"] & T^2(A \otimes X) \arrow[d, "\mu_{A\otimes X}"] \\
A \otimes TX \arrow[rr, "\theta_{A,X}"'] & & T(A \otimes X)
\end{tikzcd}$$

If the following diagram commutes, T is said to be *commutative* (and θ is a *commutative strength*):

$$\begin{tikzcd}
TA \otimes TB \arrow[r, "\theta_{TA,B}"] \arrow[d, "c"'] & T(TA \otimes B) \arrow[r, "Tc"] & T(B \otimes TA) \arrow[r, "T\theta_{B,A}"] & T^2(B \otimes A) \arrow[d, "T^2c"] \\
TB \otimes TA \arrow[d, "\theta_{TB,A}"'] & & & T^2(A \otimes B) \arrow[d, "\mu"] \\
T(TB \otimes A) \arrow[r, "Tc"'] & T(A \otimes TB) \arrow[r, "T\theta_{A,B}"'] & T^2(A \otimes B) \arrow[r, "\mu"'] & T(A \otimes B)
\end{tikzcd}$$

□

5.2 Higher-Order Extension

Since the pure higher-order sharing theory is obtained from the first-order one by adding additional term constructs and axioms, there is an obvious sound translation from the first-order theory to the higher-order theory. We show that this translation is not only sound but also conservative (faithful) and full. It is possible to prove this by a purely syntactic way (by comparing the normal forms), but here we give a simple semantic proof using a model embedding technique.

We start with a general fact on the Yoneda construction (free cocompletion) on symmetric monoidal categories [Day70]. A systematic account can be found in [PR96].

Lemma 5.2.1 Let $\mathcal{C}$, $\mathcal{D}$ be small symmetric monoidal categories, with a strict symmetric monoidal functor $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ which is identity on objects. Then there exist small symmetric monoidal categories $\bar{\mathcal{C}}$ and $\bar{\mathcal{D}}$, an identity-on-objects strict symmetric monoidal functor $\bar{\mathcal{F}} : \bar{\mathcal{C}} \to \bar{\mathcal{D}}$, together with fully faithful strict symmetric monoidal functors $i_{\mathcal{C}} : \mathcal{C} \to \bar{\mathcal{C}}$ and $i_{\mathcal{D}} : \mathcal{D} \to \bar{\mathcal{D}}$ such that the induced square commutes and $\bar{\mathcal{F}}$ has a right adjoint; moreover $\bar{\mathcal{C}}$ is symmetric monoidal closed, and $i_{\mathcal{C}}$ is dense.

Proof: Let $\bar{\mathcal{C}}$ be the presheaf category $[\mathcal{C}^{\mathrm{op}}, \mathbf{Sets}]$ and $in_{\mathcal{C}}$ be the Yoneda embedding. As well-known, $\bar{\mathcal{C}}$ is the free symmetric monoidal cocompletion of $\mathcal{C}$ and $in_{\mathcal{C}}$ is strict symmetric monoidal [Day70, IK86]. Then F extends to a strict symmetric monoidal functor $\hat{F} : \bar{\mathcal{C}} \longrightarrow [\mathcal{D}^{\mathrm{op}}, \mathbf{Sets}]$ with a right adjoint $U = [F^{\mathrm{op}}, \mathbf{Sets}]$, so that $\hat{F}$ strictly commutes with F. Although $\hat{F}$ may not be identity-on-objects, we can factorize it as $\hat{F} = J \circ \bar{F}$ so that $\bar{F} : \bar{\mathcal{C}} \longrightarrow \bar{\mathcal{D}}$ is identity-on-objects and $J : \bar{\mathcal{D}} \longrightarrow [\mathcal{D}^{\mathrm{op}}, \mathbf{Sets}]$ is fully faithful. A right adjoint of $\bar{F}$ is then given by $U \circ J : \bar{\mathcal{D}} \longrightarrow \bar{\mathcal{C}}$. ($\bar{\mathcal{C}}$ and $\bar{\mathcal{D}}$ obtained are not small, but we can cut down them to be closed and small.) $\Box$

Corollary 5.2.2 Let $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ be a cartesian-center SMC. Then there is a cartesian centrally closed SMC $\bar{\mathcal{F}} : \bar{\mathcal{C}} \to \bar{\mathcal{S}}$ with a fully faithful cartesian-center functor $(i_{\mathcal{C}}, i_{\mathcal{S}})$ from $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ to $\bar{\mathcal{F}} : \bar{\mathcal{C}} \to \bar{\mathcal{S}}$. Moreover $i_{\mathcal{C}}$ is dense. $\Box$

Theorem 5.2.3 (conservativity)
If $\Gamma \vdash M = N : (\vec{\sigma})$ is derivable in the pure higher-order acyclic sharing theory, then it is also derivable in the pure acyclic sharing theory.

Proof: Let $\mathcal{F}_{\mathbb{T}} : \mathcal{C}_{\mathbb{T}} \to \mathcal{S}_{\mathbb{T}}$ be the classifying category of the pure acyclic sharing theory. From this, we get a cartesian centrally closed SMC $\bar{\mathcal{F}}_{\mathbb{T}} : \bar{\mathcal{C}}_{\mathbb{T}} \to \bar{\mathcal{S}}_{\mathbb{T}}$ to which the classifying category fully and faithfully embeds. (This is not strict, so we need to take its strict equivalent to be more precise.) The sharing structure in $\mathcal{F}_{\mathbb{T}} : \mathcal{C}_{\mathbb{T}} \to \mathcal{S}_{\mathbb{T}}$ canonically extends to a higher-order sharing structure in $\bar{\mathcal{F}}_{\mathbb{T}} : \bar{\mathcal{C}}_{\mathbb{T}} \to \bar{\mathcal{S}}_{\mathbb{T}}$, and it is routine to see that it induces a higher-order sharing model. Then the fully faithful embedding from $\mathcal{F}_{\mathbb{T}} : \mathcal{C}_{\mathbb{T}} \to \mathcal{S}_{\mathbb{T}}$ to $\bar{\mathcal{F}}_{\mathbb{T}} : \bar{\mathcal{C}}_{\mathbb{T}} \to \bar{\mathcal{S}}_{\mathbb{T}}$ factors through the classifying category of the pure higher-order acyclic sharing theory, and this implies that the canonical translation from $\mathcal{F}_{\mathbb{T}} : \mathcal{C}_{\mathbb{T}} \to \mathcal{S}_{\mathbb{T}}$ to the classifying category of the higher-order theory is faithful. □

5.3 Notions of Computation

It has been pointed that Moggi's computational lambda calculus [Mog88] looks like higher-order graph rewriting systems. And we notice that our higher-order acyclic sharing theory is fairly similar to Moggi's calculus. We make this intuition precise by observing that Moggi's models and ours are essentially the same. This comparison can be understood as an instance of the work by Power and Robinson [PR96], where they give a reformulation of Moggi's notions of computation in terms of premonoidal categories. In this thesis we use just symmetric monoidal categories, but the story presented here is inspired from their general framework.

Definition 5.3.1 (λ_c-models [Mog88])
Let $\mathcal{C}$ be a cartesian category. A *λ_c-model over $\mathcal{C}$* is a strong monad (T, η, μ, θ) which satisfies the mono-requirement (each component of η is mono) and has Kleisli exponents, i.e. there is a chosen object $A \Rightarrow B$ for each object A and B and a natural isomorphism

$$\mathcal{C}_T(J(- \times A), -) \simeq \mathcal{C}(-, A \Rightarrow -)$$

where $\mathcal{C}_T$ is the Kleisli category of T and $J : \mathcal{C} \longrightarrow \mathcal{C}_T$ is given by $J(f) = f; \eta$. □

Definition 5.3.2 A λ_c-model is said to be *commutative* if the tensorial strength is commutative. □

Theorem 5.3.3 There is a bijective correspondence between commutative λ_c-models and faithful cartesian centrally closed SMC's.

Proof: Routine; see for instance [PR96]. □

We reproduce Moggi's computational lambda calculus (λ_c-calculus) [Mog88] (the simply typed version in [MOTW95]) as below.

Definition 5.3.4 (computational lambda calculus)
The *computational lambda calculus (λ_c-calculus)* [Mog88, MOTW95] is determined by the following data.
[Types]

$$\sigma, \tau \ldots \quad ::= \quad b \mid \sigma \Rightarrow \tau \quad \text{(where } b \text{ is a base type)}$$

[Syntactic domains]

Variables	$x, y, z \ldots$	
Raw Terms	$M, N \ldots$	$::= \ x \mid \lambda x.M \mid MN \mid \mathsf{let}\ x = M\ \mathsf{in}\ N$
Values	$V, W \ldots$	$::= \ x \mid \lambda x.M$

[Typing]

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma} \quad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \Rightarrow \tau}$$

$$\frac{\Gamma \vdash M : \sigma \Rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \quad \frac{\Gamma \vdash M : \sigma \quad \Gamma, x : \sigma \vdash N : \tau}{\Gamma \vdash \mathsf{let}\ x = M\ \mathsf{in}\ N : \tau}$$

[Axioms]

(β_v)	$(\lambda x.M)V$	$=$	$M\{V/x\}$
(η_v)	$\lambda x.(Vx)$	$=$	V
			$(x \notin FV(V))$
(let_v)	$\mathsf{let}\ x = V\ \mathsf{in}\ M$	$=$	$M\{V/x\}$
(id)	$\mathsf{let}\ x = M\ \mathsf{in}\ x$	$=$	M
(comp)	$\mathsf{let}\ y = (\mathsf{let}\ x = L\ \mathsf{in}\ M)\ \mathsf{in}\ N$	$=$	$\mathsf{let}\ x = L\ \mathsf{in}\ \mathsf{let}\ y = M\ \mathsf{in}\ N$
(let.1)	EM	$=$	$\mathsf{let}\ z = E\ \mathsf{in}\ zM$
(let.2)	VE	$=$	$\mathsf{let}\ x = E\ \mathsf{in}\ Vx$

where E ranges over non-values, i.e. applications and let-blocks. □

Definition 5.3.5 The *commutative λ_c-calculus* is obtained by assuming the following additional axiom.

$$\text{(comm)} \quad \mathsf{let}\ x = L\ \mathsf{in}\ \mathsf{let}\ y = M\ \mathsf{in}\ N \quad = \quad \mathsf{let}\ y = M\ \mathsf{in}\ \mathsf{let}\ x = L\ \mathsf{in}\ N$$

□

Since Moggi's (commutative) calculus is sound and complete for (commutative) λ_c-models, we have

Theorem 5.3.6 The pure higher-order acyclic sharing theory is a conservative extension of the commutative computational lambda calculus. □

5.4 Models of Intuitionistic Linear Logic

A model of propositional intuitionistic linear logic may be described as a symmetric monoidal adjunction between a cartesian closed category and a symmetric monoidal closed category [Bar96, BW96, Ben95, Bie95]. Such a structure induces a cartesian-center SMC, i.e. a model of the acyclic sharing theory, which implies that there is a sound interpretation from the sharing theory into the term calculus of intuitionistic type theory. We show that it is conservative, thus a linear type theory is seen as a conservative extension of the pure acyclic sharing theory. Again we avoid syntactic proof, by relating the classes of models.

Definition 5.4.1 (LNL models [Ben95])
A *linear/non-linear model* (*LNL model*) is a symmetric monoidal adjunction $\begin{tikzcd} \mathcal{C} \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] & \mathcal{S} \arrow[l, shift left, "G"] \end{tikzcd}$ where $\mathcal{C}$ is a cartesian closed category and $\mathcal{S}$ is a symmetric monoidal closed category. □

First, it is fairly easy to see that an LNL model induces a cartesian centrally closed SMC (equivalently Moggi's model, as observed in [BW96]).

Lemma 5.4.2 Let $\begin{tikzcd} \mathcal{C} \arrow[r, shift left, "F"] \arrow[r, phantom, "\perp"] & \mathcal{S} \arrow[l, shift left, "G"] \end{tikzcd}$ be an LNL model. Consider the monad $T = G \circ F$ and its Kleisli category $\mathcal{C}_T$. Then the canonical functor $J : \mathcal{C} \to \mathcal{C}_T$ is a cartesian centrally closed SMC; and $\mathcal{C}_T$ is a full subcategory of $\mathcal{S}$; and F factors through J. □

Intuitively, $\mathcal{C}_T$ corresponds to a fragment of the linear type theory in which types are of the form $!A$. Therefore a (higher-order) sharing theory can be translated into this fragment of a linear type theory (equipped with constants corresponding to operator symbols). See [Ben95, BW96] for details of the term calculus.

The conservativity of the translation is a consequence of the following construction.

Theorem 5.4.3 Given a cartesian-center SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$, there is an LNL model $\begin{tikzcd} \widehat{\mathcal{C}} \arrow[r, shift left, "\widehat{\mathcal{F}}"] \arrow[r, phantom, "\perp"] & \widehat{\mathcal{S}} \arrow[l, shift left, "U"] \end{tikzcd}$ such that there are fully faithful strict symmetric monoidal functors $j : \mathcal{C} \to \widehat{\mathcal{C}}$ and $j' : \mathcal{S} \to \widehat{\mathcal{S}}$ which satisfy $j' \circ \mathcal{F} = \widehat{\mathcal{F}} \circ j$.

Proof: Similar to Lemma 5.2.1. □

Theorem 5.4.4 The interpretation from the pure acyclic sharing theory into the LNL logic (with operator symbols of Σ) is conservative. □

Chapter 6

Models of Cyclic Sharing Theory

We have delayed introducing the models for cyclic sharing graphs until now. The main reason is that, while the models of acyclic settings are obtained by revising well known concepts from category theory, we need a relatively new notion for interpreting cyclic bindings - *traced monoidal categories* introduced by Joyal, Street and Verity [JSV96]. The notion of trace, while the concept itself goes back to the classical traces of linear maps between finite dimensional vector spaces, has originally been invented for analyzing cyclic structures arising from mathematics and physics, notably the interplay of low-dimensional topology (knot theory) and quantum groups (e.g. [RT90, Kas95]); it is then a natural idea to use this concept for modeling our cyclic graph structure too, and it does work.

In this chapter we deal with the models of cyclic sharing theories. The additional structure we require for modeling cyclic bindings is trace as mentioned above, which we review below. The construction of this chapter is therefore parallel to that for models of acyclic sharing theories (Chapter 3), with additional considerations on the trace structure induced by cyclic bindings.

6.1 Traced Monoidal Categories

The notion of trace we give here for symmetric monoidal categories is adopted from the original definition of trace for balanced monoidal categories [JS93] in [JSV96].

For ease of presentation, in this section we write as if our monoidal categories are strict (i.e. monoidal products are strictly associative and coherence isomorphisms are identities).

Definition 6.1.1 (Traced symmetric monoidal categories [JSV96])
A symmetric monoidal category $(\mathcal{C}, \otimes, I, c)$ (where c is the symmetry; $c_{X,Y}$:

$X \otimes Y \longrightarrow Y \otimes X$) is said to be *traced* if it is equipped with a natural family of functions, called a *trace*,

$$Tr^X_{A,B} : \mathcal{C}(A \otimes X, B \otimes X) \longrightarrow \mathcal{C}(A, B)$$

subject to the following three conditions.

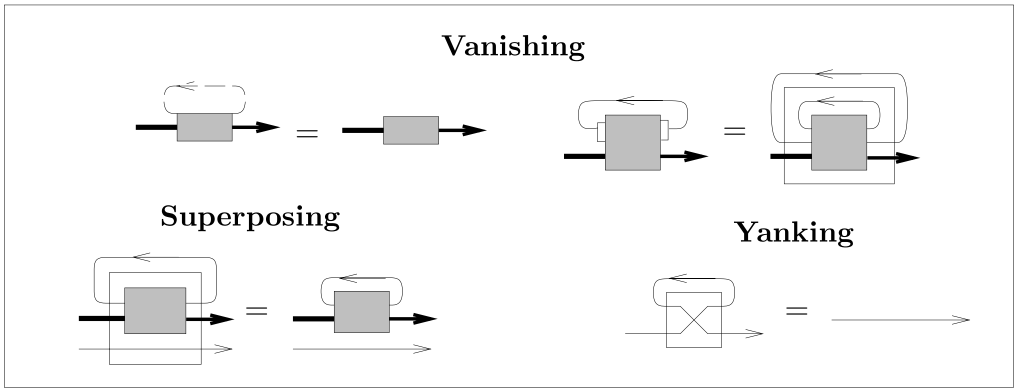
- **Vanishing:**
$$Tr^I_{A,B}(f) = f : A \to B$$
where $f : A \longrightarrow B$, and
$$Tr^{X \otimes Y}_{A,B}(f) = Tr^X_{A,B}(Tr^Y_{A \otimes X, B \otimes X}(f)) : A \longrightarrow B$$
where $f : A \otimes X \otimes Y \longrightarrow B \otimes X \otimes Y$

- **Superposing:**
$$Tr^X_{C \otimes A, C \otimes B}(id_C \otimes f) = id_C \otimes Tr^X_{A,B}(f) : C \otimes A \longrightarrow C \otimes B$$
where $f : A \otimes X \longrightarrow B \otimes X$

- **Yanking:**
$$Tr^X_{X,X}(c_{X,X}) = id_X : X \longrightarrow X \qquad \square$$

We may omit the subscripts if there is no confusion.

We present the graphical version of these axioms to help with the intuition of traced categories as categories with cycles (or feedback, reflexion). Such graphical languages for various monoidal categories have been developed in [JS91].



Note that naturality of a trace can be axiomatized as follows.

- Naturality in A (**Left Tightening**)

$$Tr^X_{A,B}((g \otimes id_X); f) = g; Tr^X_{A',B}(f) : A \longrightarrow B$$

where $f : A' \otimes X \longrightarrow B \otimes X$, $g : A \longrightarrow A'$

- Naturality in B (**Right Tightening**)

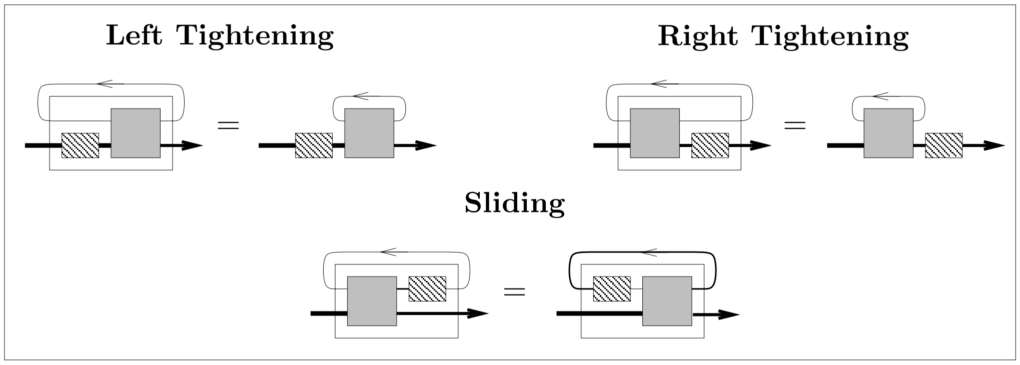
$$Tr^X_{A,B}(f; (g \otimes id_X)) = Tr^X_{A,B'}(f); g : A \longrightarrow B$$

where $f : A \otimes X \longrightarrow B' \otimes X$, $g : B' \longrightarrow B$

- Naturality in X (**Sliding**)

$$Tr^X_{A,B}(f; (id_B \otimes g)) = Tr^{X'}_{A,B}((id_A \otimes g); f) : A \longrightarrow B$$

where $f : A \otimes X \longrightarrow B \otimes X'$, $g : X' \longrightarrow X$



Remark 6.1.2 The axiom **Superposing** is slightly simplified from the original version in [JSV96]

$$Tr^X_{A\otimes C,B\otimes D}((id_A \otimes c_{C,X}); (f \otimes g); (id_B \otimes c_{X,D})) = Tr^X_{A,B}(f) \otimes g$$

where $f : A \otimes X \longrightarrow B \otimes X$, $g : C \longrightarrow D$. Assuming axioms **Left** & **Right Tightenings**, ours is derivable from this original one, and vice versa. $\Box$

Example 6.1.3 A *compact closed category* [Kel72, KL80] is a symmetric monoidal category with a contravariant endofunctor $(-)^*$ and natural transformations $\eta_A : I \to A \otimes A^*$ and $\epsilon_A : A^* \otimes A \to I$ such that

$$A \xrightarrow{r_A^{-1}} I \otimes A \xrightarrow{\eta_A \otimes A} (A \otimes A^*) \otimes A \xrightarrow{a_{A,A^*,A}} A \otimes (A^* \otimes A) \xrightarrow{A \otimes \epsilon_A} A \otimes I \xrightarrow{l_A} A$$

$$A^* \xrightarrow{l_{A^*}^{-1}} A^* \otimes I \xrightarrow{A^* \otimes \eta_A} A^* \otimes (A \otimes A^*) \xrightarrow{a^{-1}_{A^*,A,A^*}} (A^* \otimes A) \otimes A^* \xrightarrow{\epsilon_A \otimes A^*} I \otimes A^* \xrightarrow{r_{A^*}} A^*$$

agree with identity arrows. Such a category is closed, with exponents $[A, B]$ given by $A^* \otimes B$. In [JSV96], it is shown that any compact closed category is traced, for instance the category of sets and binary relations, and the category of finite dimensional vector spaces (see examples in the next chapter). Moreover, the structure theorem (which gives a more general relation between traced balanced monoidal categories and tortile monoidal categories [Shu94]) in [JSV96] tells us that any traced symmetric monoidal category can be fully and faithfully embedded into a compact closed category (which can be obtained by a simple fraction construction). This fact, however, does not imply that the usage of traced categories is the same as that of compact closed categories. For the study of cyclic data structures, we find traced categories more useful than compact closed categories, as the latter seems to be too strong for our purpose to model cyclic structures (rather than dualities which we do not need at least a priori). In particular, while we will see that there are many interesting traced cartesian categories, a compact closed category whose monoidal product is cartesian is trivial (because, for such $\mathcal{C}$, we have $\mathcal{C}(A, B) \simeq \mathcal{C}(A \otimes B^*, 1)$). □

Example 6.1.4 Examples which are not compact closed include the category of sets and partial functions with coproduct as monoidal product, and the category of sets and binary relations with biproduct as monoidal product (described in detail in [JSV96]). More examples will be introduced in this and next chapters. □

We also refer Abramsky's survey [Abr96a] for some computer science oriented examples, especially some models related to Girard's Geometry of Interaction [Gir89].

Let us briefly answer a few frequently-asked-questions on the axiomatization of traces, as this might be helpful for avoiding some misunderstanding.

Fact. $Tr^X id_X = id_I$ is not always true. □

This condition is true for all traced cartesian categories and also for many other traced categories, for instance the category of sets and binary relations, but we also have lots of counterexamples - a basic one is the category of finite dimensional vector spaces and linear maps, which is the prototypical traced monoidal category (traced in the very classical sense), where $Tr^X id_X$ is the function which multiplies the dimension of X, and it is therefore not the identity id_I unless X's dimension is 1.

A reflexive action calculus (to be introduced in Chapter 8) does not satisfy this condition. However, as suggested and discussed in [Mil94b, Mif96], assuming this condition may add some desirable strength to the calculus, since an action (process) of the form $Tr^X id_X$ seems to have no computational significance and hence can be identified with the "empty action" id_I.

Fact. A trace may not be functorial – that is, $Tr^X(f;g) = Tr^X f; Tr^X g$ does not hold in general. □

Here is evidence that an interesting trace is unlikely to be functorial:

Proposition 6.1.5 For any traced symmetric monoidal category, the functoriality condition

$$Tr^X(f;g) = Tr^X f; Tr^X g \quad \text{for any} \quad X, f, g$$

is true if and only if

$$c_{X,X} = id_{X\otimes X} \quad \text{for any} \quad X.$$

<u>Proof:</u> See Appendix. □

That is, if a symmetric monoidal category has a functorial trace, the symmetries of the form $c_{X,X}$ must be the identities – not many nontrivial examples seem to satisfy such a condition.

Finally we shall mention an observation by Plotkin:

Fact. Traced symmetric monoidal categories are "cancellable", in the sense that if $f \otimes id_{X\otimes X} = g \otimes id_{X\otimes X}$ then $f \otimes id_X = g \otimes id_X$. □

This immediately follows from $f \otimes id_X = Tr^X((f \otimes id_{X\otimes X}); (id \otimes c_{X,X}))$. As there are symmetric monoidal categories which are not cancellable, we have

Theorem 6.1.6 (Plotkin)
There are symmetric monoidal categories to which we cannot add traces freely without causing a collapse. In other words, there are symmetric monoidal categories which cannot be faithfully embedded into traced monoidal categories by strong symmetric monoidal functors. □

Plotkin conjectures that cancellable symmetric monoidal categories (which include all cartesian categories as well as all cartesian-center SMC's) faithfully embed into traced categories.

Definition 6.1.7 (traced functors [JSV96])
A strong symmetric monoidal functor (F, m, m_I) between traced symmetric monoidal categories is *traced* if $F(Tr^X_{A,B}(f)) = Tr^{FX}_{FA,FB}(m; Ff; m^{-1})$ holds for any $f : A \otimes X \to B \otimes X$. $\Box$

6.2 Cyclic Sharing Models

Definition 6.2.1 (cartesian-center traced SMC)
A *cartesian-center traced symmetric monoidal category* (*cartesian-center traced SMC*) is a cartesian-center SMC whose symmetric monoidal category part is traced. $\Box$

Definition 6.2.2 (cartesian-center traced functors)
A *cartesian-center traced functor* between cartesian-center traced categories $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ and $\mathcal{F}' : \mathcal{C}' \to \mathcal{S}'$ is a cartesian-center functor (Φ, Ψ) between them such that Ψ is traced (since Ψ is strict, we require it to preserve the trace on the nose). $\Box$

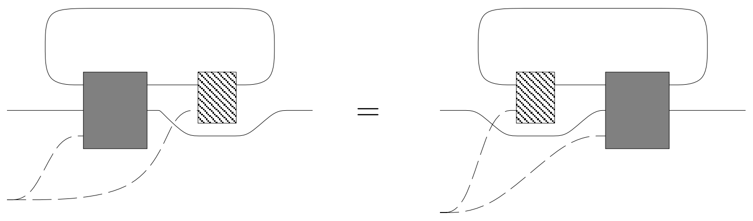
We write **CcTrSMC** for the 2-category of small cartesian-center traced symmetric monoidal categories, cartesian-center traced functors and cartesian-center natural transformations.

We again state that the simple slice construction (page 50) preserves our structure:

Lemma 6.2.3 Let $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ be a cartesian-center traced SMC and A be an object of $\mathcal{C}$ (hence $\mathcal{S}$). Then the Kleisli category $\mathcal{C}//A$ of the comonad $A \times (-)$ on $\mathcal{C}$ is a cartesian category; the Kleisli category $\mathcal{S}//A$ of the comonad $A \otimes (-)$ on $\mathcal{S}$ is a traced symmetric monoidal category; and $\mathcal{F}$ induces an identity-on-objects strict symmetric monoidal functor from $\mathcal{C}//A$ to $\mathcal{S}//A$ (for which we write $\mathcal{F}//A$). Therefore $\mathcal{F}//A : \mathcal{C}//A \to \mathcal{S}//A$ is a cartesian-center traced SMC.

Sketch of the proof: The only new point over Lemma 3.2.7 is that $\mathcal{S}//A$ is traced. Most axioms are verified easily, except **Sliding** for which we give the calculation below. We want to show $Tr^X(f; (C \otimes g)) = Tr^Y((B \otimes g); f)$ for $f : B \otimes X \to C \otimes Y$

and $g : Y \to X$ in $\mathcal{S}//A$, hence to show that $Tr^X((\mathcal{F}\Delta_A \otimes B \otimes X); (A \otimes f); (c_{A,C} \otimes Y); (C \otimes g))$ equals to $Tr^Y((\mathcal{F}\Delta_A \otimes B \otimes Y); (A \otimes c_{A,B} \otimes Y); (A \otimes B \otimes g); f)$ for $f : A \otimes B \otimes X \to C \otimes Y$ and $g : A \otimes Y \to X$ in $\mathcal{S}$. The following picture may help with the intuition:



$$
\begin{array}{lll}
 & Tr^X((\mathcal{F}\Delta_A \otimes B \otimes X); (A \otimes f); (c_{A,C} \otimes Y); (C \otimes g)) & \\
= & Tr^X((\mathcal{F}\Delta_A \otimes B \otimes X); (A \otimes f); (c_{A,C} \otimes Y); (C \otimes g)) & \\
= & Tr^{A \otimes Y}((\mathcal{F}\Delta_A \otimes B \otimes g); (A \otimes f); (c_{A,C} \otimes Y)) & \text{Sliding} \\
= & Tr^A(Tr^Y((\mathcal{F}\Delta_A \otimes B \otimes g); (A \otimes f); (c_{A,C} \otimes Y))) & \text{Vanishing} \\
= & Tr^A((\mathcal{F}\Delta_A \otimes B \otimes A); Tr^Y((A \otimes A \otimes B \otimes g); (A \otimes f)); c_{A,C}) & \text{L.\& R. T.} \\
= & Tr^A((\mathcal{F}\Delta_A \otimes B \otimes A); (A \otimes Tr^Y((A \otimes B \otimes g); f)); c_{A,C}) & \text{L.\& R. T.} \\
= & Tr^A((\mathcal{F}\Delta_A \otimes B \otimes A); c_{A,A \otimes B \otimes A}; (Tr^Y((A \otimes B \otimes g); f) \otimes A)) & \\
= & (\mathcal{F}\Delta_A \otimes B); Tr^A(c_{A,A \otimes B \otimes A}); Tr^Y((A \otimes B \otimes g); f) & \text{L.\& R. T.} \\
= & (\mathcal{F}\Delta_A \otimes B); Tr^A((c_{A,A \otimes B} \otimes A); (A \otimes B \otimes c_{A,A})); Tr^Y((A \otimes B \otimes g); f) & \\
= & (\mathcal{F}\Delta_A \otimes B); c_{A,A \otimes B}; Tr^A(A \otimes B \otimes c_{A,A}); Tr^Y((A \otimes B \otimes g); f) & \text{Left Tight.} \\
= & (\mathcal{F}\Delta_A \otimes B); c_{A,A \otimes B}; (A \otimes B \otimes Tr^A(c_{A,A})); Tr^Y((A \otimes B \otimes g); f) & \text{Superpose.} \\
= & (\mathcal{F}\Delta_A \otimes B); c_{A,A \otimes B}; Tr^Y((A \otimes B \otimes g); f) & \text{Yanking} \\
= & (\mathcal{F}\Delta_A \otimes B); (c_{A,A} \otimes B); (A \otimes c_{A,B}); Tr^Y((A \otimes B \otimes g); f) & \\
= & (\mathcal{F}\Delta_A \otimes B); (A \otimes c_{A,B}); Tr^Y((A \otimes B \otimes g); f) & \\
= & Tr^Y((\mathcal{F}\Delta_A \otimes B \otimes Y); (A \otimes c_{A,B} \otimes Y); (A \otimes B \otimes g); f) & \text{Left Tight.}
\end{array}
$$

$\square$

Corollary 6.2.4 If $\mathcal{C}$ is a traced cartesian category and A is an object of $\mathcal{C}$, then $\mathcal{C}//A$ is also a traced cartesian category. $\square$

These results will be used in the proofs of Theorem 7.1.1 and Theorem 7.2.1.

Now we proceed to give the semantic interpretation of cyclic sharing theories in cartesian-center traced SMC's. We repeat the same pattern with Chapter 3, except the treatment of the letrec-bindings.

Definition 6.2.5 (cyclic sharing structures)
A *cyclic sharing structure* over an S-sorted signature Σ is a sharing structure (Definition 3.2.9) in a strict cartesian-center traced SMC. $\square$

Given a cyclic sharing structure $[\![-]\!]$ in a cartesian-center traced SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$, we define $[\![x_1 : \sigma_1, \ldots, x_m : \sigma_m \vdash M : (\tau_1, \ldots, \tau_n)]\!] : [\![(\sigma_1, \ldots, \sigma_m)]\!] \to [\![(\tau_1, \ldots, \tau_n)]\!]$ in $\mathcal{S}$ for each well-typed term $x_1 : \sigma_1, \ldots, x_m : \sigma_m \vdash M : (\tau_1, \ldots, \tau_n)$ as follows, by induction on the typing rules.

$$\begin{array}{lcl}
[\![\Gamma, x : \sigma \vdash x : (\sigma)]\!] & = & \mathcal{F}(\pi') \\
[\![\Gamma \vdash F(M) : (\vec{\tau})]\!] & = & [\![\Gamma \vdash M : (\vec{\sigma})]\!]; [\![F]\!] \\
[\![\Gamma \vdash 0 : ()]\!] & = & \mathcal{F}(!) \\
[\![\Gamma \vdash M \otimes N : (\vec{\sigma}, \vec{\tau})]\!] & = & \mathcal{F}(\Delta); ([\![\Gamma \vdash M : (\vec{\sigma})]\!] \otimes [\![\Gamma \vdash N : (\vec{\tau})]\!]) \\
[\![\Gamma \vdash \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]\!] & = & \mathcal{F}(\Delta); (id \otimes Tr^{[\![(\vec{\sigma})]\!]}([\![\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\sigma})]\!]; \mathcal{F}(\Delta))); \\
 & & \quad [\![\Gamma, \vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})]\!] \\
[\![\Gamma, x' : \sigma', x : \sigma, \Gamma' \vdash M : (\vec{\tau})]\!] & = & (id \otimes \mathcal{F}(c) \otimes id); [\![\Gamma, x : \sigma, x' : \sigma', \Gamma' \vdash M : (\vec{\tau})]\!]
\end{array}$$

Lemma 6.2.6 Let $[\![-]\!]$ be a cyclic sharing structure. Then $[\![\vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})]\!] = [\![\vec{y} : \vec{\sigma} \vdash M\{\vec{y}/\vec{x}\} : (\vec{\tau})]\!]$ where the x's and y's are disjoint.

Proof: Induction on the construction of M. □

Definition 6.2.7 (cyclic sharing models)
A *cyclic sharing model* of a cyclic sharing theory is a cyclic sharing structure $[\![-]\!]$ in a strict cartesian-center traced SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ such that $[\![\Gamma \vdash M : (\vec{\sigma})]\!] = [\![\Gamma \vdash N : (\vec{\sigma})]\!]$ for each axiom $\Gamma \vdash M = N : (\vec{\sigma})$ of the theory. □

Theorem 6.2.8 (soundness)
Let $[\![-]\!]$ be a sharing model of a sharing theory. If $\Gamma \vdash M = N : (\vec{\sigma})$ is derivable in the theory, then $[\![\Gamma \vdash M : (\vec{\sigma})]\!] = [\![\Gamma \vdash N : (\vec{\sigma})]\!]$.

Proof: Similar to the case of acyclic theories, but the calculation becomes considerably complicated because of the trace axioms. We shall check the axioms in Definition 2.3.3 (page 34).

- (σ_{var}): Put $p = [\![\Gamma, \vec{y} : \vec{\sigma}' \vdash z : (\sigma)]\!]$, $f = [\![\Gamma, x : \sigma, \vec{y} : \vec{\sigma}' \vdash M : (\vec{\sigma}')]\!]$ and $g = [\![\Gamma, x : \sigma, \vec{y} : \vec{\sigma}' \vdash N : (\vec{\tau})]\!]$. Also define $f' = [\![\Gamma, \vec{y} : \vec{\sigma}' \vdash M\{z/x\} : (\vec{\sigma}')]\!]$ and $g' = [\![\Gamma, \vec{y} : \vec{\sigma}' \vdash N\{z/x\} : (\vec{\tau})]\!]$. By induction on the constructions of M and N, we can show that $f' = (\mathcal{F}\Delta \otimes \mathcal{F}\Delta); (id \otimes p \otimes id); f$ and $g' = (\mathcal{F}\Delta \otimes \mathcal{F}\Delta); (id \otimes p \otimes id); g$ as in the proof of (σ_{var}) for the acyclic case. Then

$$\begin{array}{rl}
 & [\![\Gamma \vdash \mathsf{letrec}\ (x, \vec{y})\ \mathsf{be}\ z \otimes M\ \mathsf{in}\ N : (\vec{\tau})]\!] \\
 = & \mathcal{F}\Delta; (id \otimes Tr^{[\![(\sigma, \vec{\sigma}')]\!]}((\mathcal{F}\Delta \otimes id \otimes \mathcal{F}\Delta); (id \otimes \mathcal{F}c \otimes id); (p \otimes f); \mathcal{F}\Delta)); g
\end{array}$$

$$\begin{array}{rl}
 & [\![\Gamma \vdash \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ M\{z/x\}\ \mathsf{in}\ N\{z/x\} : (\vec{\tau})]\!] \\
 = & \mathcal{F}\Delta; (id \otimes Tr^{[\![\vec{\sigma}']\!]}(f'; \mathcal{F}\Delta)); g'.
\end{array}$$

So it suffices to show

$$\begin{array}{rl} & Tr^{[\![(\sigma,\vec{\sigma}')]\!]}((\mathcal{F}\Delta \otimes id \otimes \mathcal{F}\Delta);(id \otimes \mathcal{F}c \otimes id);(p \otimes f);\mathcal{F}\Delta) \\ = & \mathcal{F}\Delta;(id \otimes Tr^{[\![\vec{\sigma}']\!]}(f';\mathcal{F}\Delta);\mathcal{F}\Delta);(p \otimes id). \end{array}$$

$$\begin{array}{rll} & LHS & \\ = & Tr(Tr((\mathcal{F}\Delta \otimes id \otimes \mathcal{F}\Delta);(id \otimes \mathcal{F}c \otimes id);(p \otimes f);\mathcal{F}\Delta)) & \text{Vanish.} \\ = & Tr(\mathcal{F}\Delta;(p;\mathcal{F}\Delta \otimes id);(id \otimes (\mathcal{F}c \otimes id);f;\mathcal{F}\Delta)) & \text{Sld., R.T, Yank.} \\ = & Tr(\mathcal{F}\Delta;(\mathcal{F}\Delta(p \otimes p) \otimes id);(id \otimes (\mathcal{F}c \otimes id);f;\mathcal{F}\Delta)) & \\ = & RHS & \text{R.T.} \end{array}$$

- (id) is the same as in the acyclic case.

- (ass$_1$): Put $f = [\![\Gamma, \vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash L : (\vec{\sigma}')]\!]$, $g = [\![\Gamma, \vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}' \vdash M : (\vec{\sigma})]\!]$ and $h = [\![\Gamma, \vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})]\!]$. Then

$$\begin{array}{rl} & [\![\Gamma \vdash \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ (\mathsf{letrec}\,(\vec{y})\,\mathsf{be}\,L\,\mathsf{in}\,M)\ \mathsf{in}\ N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta;(id \otimes Tr^{[\![(\vec{\sigma})]\!]}(\mathcal{F}\Delta;(id \otimes Tr^{[\![(\vec{\sigma}')]\!]}(f;\mathcal{F}\Delta));g;\mathcal{F}\Delta));h \end{array}$$

$$\begin{array}{rl} & [\![\Gamma \vdash \mathsf{letrec}\ (\vec{x},\vec{y})\ \mathsf{be}\ M \otimes L\ \mathsf{in}\ N :: (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta;(id \otimes Tr^{[\![(\vec{\sigma},\vec{\sigma}')]\!]}(\mathcal{F}\Delta;(g \otimes f);\mathcal{F}\Delta));\mathcal{F}\pi;h. \end{array}$$

We shall show

$$Tr^{[\![(\vec{\sigma})]\!]}(\mathcal{F}\Delta;(id \otimes Tr^{[\![(\vec{\sigma}')]\!]}(f;\mathcal{F}\Delta);g;\mathcal{F}\Delta)) = Tr^{[\![(\vec{\sigma},\vec{\sigma}')]\!]}(\mathcal{F}\Delta;(g \otimes f);\mathcal{F}\Delta);\mathcal{F}\pi.$$

$$\begin{array}{rll} & RHS & \\ = & Tr(Tr(\mathcal{F}\Delta;(g \otimes f);\mathcal{F}\Delta));\mathcal{F}\pi & \text{Vanishing} \\ = & Tr(\mathcal{F}\Delta;Tr((id \otimes \mathcal{F}\Delta);(id \otimes \mathcal{F}c \otimes id);(id \otimes f;\mathcal{F}\Delta)); & \\ & (g;\mathcal{F}\Delta \otimes id));\pi & \text{L.T.\&R.T.} \\ = & Tr(\mathcal{F}\Delta;(id \otimes Tr((id \otimes \mathcal{F}\Delta);(\mathcal{F}c \otimes id);(id \otimes f;\mathcal{F}\Delta))); & \\ & (g;\mathcal{F}\Delta \otimes id);\pi & \text{Superposing} \\ = & Tr(\mathcal{F}\Delta;(id \otimes Tr((id \otimes \mathcal{F}\Delta);(\mathcal{F}c \otimes id);(id \otimes f)));g;\mathcal{F}\Delta) & \text{R.T.} \\ = & Tr(\mathcal{F}\Delta;(id \otimes Tr((\mathcal{F}c \otimes id);(id \otimes f);(id \otimes \mathcal{F}\Delta)));g;\mathcal{F}\Delta) & \text{Sliding} \\ = & LHS & \text{L.T, \&Yank.} \end{array}$$

- (ass$_2$) is the same as in the acyclic theory.

- $\otimes_i$ are the same as in the acyclic theory.

- (perm): Put $f = [\![\Gamma, \vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}', \vec{z} : \vec{\sigma}'' \vdash M_1 \otimes M_2 \otimes M_3 : (\vec{\sigma}, \vec{\sigma}', \vec{\sigma}'')]\!]$ and $g = [\![\Gamma, \vec{y} : \vec{\sigma}', \vec{x} : \vec{\sigma}, \vec{z} : \vec{\sigma}'' \vdash M_2 \otimes M_1 \otimes M_3 : (\vec{\sigma}', \vec{\sigma}, \vec{\sigma}'')]\!]$. Then $f = (id \otimes \mathcal{F}c \otimes id);g;(\mathcal{F}c \otimes id)$, and by Sliding and Right Tightening $Tr^{[\![(\vec{\sigma},\vec{\sigma}',\vec{\sigma}'')]\!]}(f;\mathcal{F}\Delta) = Tr^{[\![(\vec{\sigma}',\vec{\sigma},\vec{\sigma}'')]\!]}(g;\mathcal{F}\Delta);(\mathcal{F}c \otimes id)$. Now (perm) is proved by

$$\begin{array}{rl} & [\![\Gamma \vdash \mathsf{letrec}\ (\vec{x},\vec{y},\vec{z})\ \mathsf{be}\ M_1 \otimes M_2 \otimes M_3\ \mathsf{in}\ N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta;(id \otimes Tr^{[\![(\vec{\sigma},\vec{\sigma}',\vec{\sigma}'')]\!]}(f;\mathcal{F}\Delta));[\![\Gamma, \vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}', \vec{z} : \vec{\sigma}'' \vdash N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta;(id \otimes Tr^{[\![(\vec{\sigma}',\vec{\sigma},\vec{\sigma}'')]\!]}(g;\mathcal{F}\Delta);(\mathcal{F}c \otimes id));[\![\Gamma, \vec{x} : \vec{\sigma}, \vec{y} : \vec{\sigma}', \vec{z} : \vec{\sigma}'' \vdash N : (\vec{\tau})]\!] \\ = & \mathcal{F}\Delta;(id \otimes Tr^{[\![(\vec{\sigma}',\vec{\sigma},\vec{\sigma}'')]\!]}(g;\mathcal{F}\Delta));[\![\Gamma, \vec{y} : \vec{\sigma}', \vec{x} : \vec{\sigma}, \vec{z} : \vec{\sigma}'' \vdash N : (\vec{\tau})]\!] \\ = & [\![\Gamma \vdash \mathsf{letrec}\ (\vec{y},\vec{x},\vec{z})\ \mathsf{be}\ M_2 \otimes M_1 \otimes M_3\ \mathsf{in}\ N : (\vec{\tau})]\!] \end{array}$$

- (subst) is the same as in the acyclic case.

□

For a cyclic sharing theory $\mathbb{T}$ (determined by a set of axioms) over a signature Σ, we write $\mathbf{CSharingMod}(\mathbb{T}, (\mathcal{F} : \mathcal{C} \to \mathcal{S}))$ for the category of $\mathbb{T}$'s cyclic sharing models in a cartesian-center traced SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ and the homomorphisms between models.

6.3 The Classifying Category

Proposition 6.3.1 Given a cyclic sharing theory $\mathbb{T}$ over Σ, there is a strict cartesian-center traced SMC $\mathcal{F}_{\mathbb{T}} : \mathcal{C}_{\mathbb{T}} \to \mathcal{S}_{\mathbb{T}}$ obtained exactly in the same way as we did for constructing the classifying category of an acyclic sharing theory. The only difference is that let-bindings are replaced by letrec-bindings

$$[\Gamma \vdash M : (\vec{\sigma})]; [\vec{x} : \vec{\sigma} \vdash N : (\vec{\tau})] \;=\; [\Gamma \vdash \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\tau})]$$

which are powerful enough to determine a trace structure on $\mathcal{S}_{\mathbb{T}}$ as

$$Tr^{(\vec{\sigma})}([\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\tau}, \vec{\sigma})]) \;=\; [\Gamma \vdash \mathsf{letrec}\ (\vec{y}, \vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{y} : (\vec{\tau})].$$

Proof: We need to verify the axioms for traces.

- Vanishing:

$$\begin{array}{lll}
 & Tr^{(\vec{\sigma})}(Tr^{(\vec{\sigma}')}([\Gamma, \vec{x} : \vec{\sigma}, \vec{x}' : \vec{\sigma'} \vdash M : (\vec{\tau}, \vec{\sigma}, \vec{\sigma}')])) & \\
= & Tr^{(\vec{\sigma})}([\Gamma, \vec{x} : \vec{\sigma} \vdash\ \mathsf{letrec}\ (\vec{y}, \vec{z}, \vec{x}')\ \mathsf{be}\ M\ \mathsf{in}\ \vec{y} \otimes \vec{z} : (\vec{\tau}, \vec{\sigma})]) & \\
= & [\Gamma \vdash \mathsf{letrec}\ (\vec{y}', \vec{x})\ \mathsf{be}\ (\mathsf{letrec}\ (\vec{y}, \vec{z}, \vec{x}')\ \mathsf{be}\ M\ \mathsf{in}\ \vec{y} \otimes \vec{z})\ \mathsf{in}\ \vec{y}' : (\vec{\tau})] & \\
= & [\Gamma \vdash \mathsf{letrec}\ (\vec{y}', \vec{x}, \vec{y}, \vec{z}, \vec{x}')\ \mathsf{be}\ \vec{y} \otimes \vec{z} \otimes M\ \mathsf{in}\ \vec{y}' : (\vec{\tau})] & (\mathrm{ass}_1) \\
= & [\Gamma \vdash \mathsf{letrec}\ (\vec{x}, \vec{y}, \vec{z}, \vec{x}')\ \mathsf{be}\ \vec{z} \otimes M\ \mathsf{in}\ \vec{y} : (\vec{\tau})] & (\sigma_{\mathrm{var}}) \\
= & [\Gamma \vdash \mathsf{letrec}\ (\vec{y}, \vec{z}, \vec{x}')\ \mathsf{be}\ M\{\vec{z}/\vec{x}\}\ \mathsf{in}\ \vec{y} : (\vec{\tau})] & (\sigma_{\mathrm{var}}) \\
= & [\Gamma \vdash \mathsf{letrec}\ (\vec{y}, \vec{x}, \vec{x}')\ \mathsf{be}\ M\ \mathsf{in}\ \vec{y} : (\vec{\tau})] & (\alpha) \\
= & Tr^{(\vec{\sigma}, \vec{\sigma}')}([\Gamma, \vec{x} : \vec{\sigma}, \vec{x}' : \vec{\sigma'} \vdash M : (\vec{\tau}, \vec{\sigma}, \vec{\sigma}')]) &
\end{array}$$

  The another vanishing axiom is trivial.

- Superposing:

$$\begin{array}{lll}
 & Tr^{(\vec{\sigma})}(id_{(\tau)} \otimes [\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\sigma}', \vec{\sigma})]) & \\
= & Tr^{(\vec{\sigma})}([\vec{y} : \vec{\tau}, \Gamma, \vec{x} : \vec{\sigma} \vdash \vec{y} \otimes M : (\vec{y}, \vec{\sigma}', \vec{\sigma})]) & \\
= & [\vec{y} : \vec{\tau}, \Gamma \vdash \mathsf{letrec}\ (\vec{y}', \vec{x}', \vec{x})\ \mathsf{be}\ \vec{y} \otimes M\ \mathsf{in}\ \vec{y}' \otimes \vec{x}' : (\vec{\tau}, \vec{\sigma}')] & \\
= & [\vec{y} : \vec{\tau}, \Gamma \vdash \mathsf{letrec}\ (\vec{x}', \vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{y} \otimes \vec{x}' : (\vec{\tau}, \vec{\sigma}')] & (\sigma_{\mathrm{var}}) \\
= & [\vec{y} : \vec{\tau}, \Gamma \vdash \vec{y} \otimes (\mathsf{letrec}\ (\vec{x}', \vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{x}') : (\vec{\tau}, \vec{\sigma}')] & (\otimes_1) \\
= & id_{(\vec{\tau})} \otimes [\Gamma \vdash \mathsf{letrec}\ (\vec{x}', \vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{x}' : (\vec{\sigma}')] & \\
= & id_{(\vec{\tau})} \otimes Tr^{(\vec{\sigma})}([\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\sigma}', \vec{\sigma})]) &
\end{array}$$

- Yanking:

$$\begin{array}{lll}
 & Tr^{(\vec{\sigma})}(c_{(\vec{\sigma}),(\vec{\sigma})}) & \\
= & Tr^{(\vec{\sigma})}([\vec{x}:\vec{\sigma},\vec{y}:\vec{\sigma} \vdash \vec{y}\otimes\vec{x} : (\vec{\sigma},\vec{\sigma})]) & \\
= & [\vec{x}:\vec{\sigma} \vdash \mathsf{letrec}\ (\vec{z},\vec{y})\ \mathsf{be}\ \vec{y}\otimes\vec{x}\ \mathsf{in}\ \vec{z} : (\vec{\sigma})] & \\
= & [\vec{x}:\vec{\sigma} \vdash \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ \vec{x}\ \mathsf{in}\ \vec{y} : (\vec{\sigma})] & (\sigma_{\mathrm{var}}) \\
= & [\vec{x}:\vec{\sigma} \vdash \vec{x} : (\vec{\sigma})] & (\mathrm{id}) \\
= & id_{(\vec{\sigma})} &
\end{array}$$

- Left Tightening:

$$\begin{array}{lll}
 & Tr^{(\vec{\sigma})}(([\Gamma\vdash M:(\vec{\tau})]\otimes id_{(\vec{\sigma})});[\vec{y}:\vec{\tau},\vec{x}:\vec{\sigma}\vdash N:(\vec{\sigma}',\vec{\sigma})]) & \\
= & Tr^{(\vec{\sigma})}([\Gamma,\vec{z}:\vec{\sigma}\vdash M\otimes\vec{z}:(\vec{\tau},\vec{\sigma})];[\vec{y}:\vec{\tau},\vec{x}:\vec{\sigma}\vdash N:(\vec{\sigma}',\vec{\sigma})]) & \\
= & Tr^{(\vec{\sigma})}([\Gamma,\vec{z}:\vec{\sigma}\vdash \mathsf{letrec}\ (\vec{y},\vec{x})\ \mathsf{be}\ M\otimes\vec{z}\ \mathsf{in}\ N:(\vec{\sigma}',\vec{\sigma})]) & \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{u},\vec{z})\ \mathsf{be}\ (\mathsf{letrec}\ (\vec{y},\vec{x})\ \mathsf{be}\ M\otimes\vec{z}\ \mathsf{in}\ N)\ \mathsf{in}\ \vec{u}:(\vec{\sigma}')] & \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{u},\vec{z},\vec{y},\vec{x})\ \mathsf{be}\ N\otimes M\otimes\vec{z}\ \mathsf{in}\ \vec{u}:(\vec{\sigma}')] & (\mathrm{ass}_1) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{x},\vec{u},\vec{z},\vec{y})\ \mathsf{be}\ \vec{z}\otimes N\otimes M\ \mathsf{in}\ \vec{u}:(\vec{\sigma}')] & (\mathrm{perm}) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{u},\vec{z},\vec{y})\ \mathsf{be}\ N\{\vec{z}/\vec{x}\}\otimes M\ \mathsf{in}\ \vec{u}:(\vec{\sigma}')] & (\sigma_{\mathrm{var}}) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{y},\vec{u},\vec{z})\ \mathsf{be}\ M\otimes N\{\vec{z}/\vec{x}\}\ \mathsf{in}\ \vec{u}:(\vec{\sigma}')] & (\mathrm{perm}) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{letrec}\ (\vec{u},\vec{z})\ \mathsf{be}\ N\{\vec{z}/\vec{x}\}\ \mathsf{in}\ \vec{u}:(\vec{\sigma}')] & (\mathrm{ass}_2) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{letrec}\ (\vec{u},\vec{x})\ \mathsf{be}\ N\ \mathsf{in}\ \vec{u}:(\vec{\sigma}')] & (\alpha) \\
= & [\Gamma\vdash M:(\vec{\tau})];[\vec{y}:\vec{\tau}\vdash \mathsf{letrec}\ (\vec{u},\vec{x})\ \mathsf{be}\ N\ \mathsf{in}\ \vec{u}:(\vec{\sigma}')] & \\
= & [\Gamma\vdash M:(\vec{\tau})];Tr^{(\vec{\sigma})}([\vec{y}:\vec{\tau},\vec{x}:\vec{\sigma}\vdash N:(\vec{\sigma}',\vec{\sigma})]) &
\end{array}$$

- Right Tightening:

$$\begin{array}{lll}
 & Tr^{(\vec{\sigma})}([\Gamma,\vec{x}:\vec{\sigma}\vdash M:(\vec{\tau},\vec{\sigma})];([\vec{y}:\vec{\tau}\vdash N:(\vec{\tau}')]\otimes id_{(\vec{\sigma})})) & \\
= & Tr^{(\vec{\sigma})}([\Gamma,\vec{x}:\vec{\sigma}\vdash M:(\vec{\tau},\vec{\sigma})];[\vec{y}:\vec{\tau},\vec{z}:\vec{\sigma}\vdash N\otimes\vec{z}:(\vec{\tau}',\vec{\sigma})]) & \\
= & Tr^{(\vec{\sigma})}([\Gamma,\vec{x}:\vec{\sigma}\vdash \mathsf{letrec}\ (\vec{y},\vec{z})\ \mathsf{be}\ M\ \mathsf{in}\ N\otimes\vec{z}:(\vec{\tau}',\vec{\sigma})]) & \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{u},\vec{x})\ \mathsf{be}\ (\mathsf{letrec}\ (\vec{y},\vec{z})\ \mathsf{be}\ M\ \mathsf{in}\ N\otimes\vec{z})\ \mathsf{in}\ \vec{u}:(\vec{\tau}')] & \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{u},\vec{x},\vec{y},\vec{z})\ \mathsf{be}\ N\otimes\vec{z}\otimes M\ \mathsf{in}\ \vec{u}:(\vec{\tau}')] & (\mathrm{ass}_1) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{x},\vec{y},\vec{z},\vec{u})\ \mathsf{be}\ \vec{z}\otimes M\otimes N\ \mathsf{in}\ \vec{u}:(\vec{\tau}')] & (\mathrm{perm}) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{y},\vec{z},\vec{u})\ \mathsf{be}\ M\otimes N\{\vec{z}/\vec{x}\}\ \mathsf{in}\ \vec{u}:(\vec{\tau}')] & (\sigma_{\mathrm{var}}) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{y},\vec{x},\vec{u})\ \mathsf{be}\ M\otimes N\ \mathsf{in}\ \vec{u}:(\vec{\tau}')] & (\alpha) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{y},\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \mathsf{letrec}\ (\vec{u})\ \mathsf{be}\ N\ \mathsf{in}\ \vec{u}:(\vec{\tau}')] & (\mathrm{ass}_2) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{y},\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N:(\vec{\tau}')] & (\mathrm{id}) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{v},\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N\{\vec{v}/\vec{y}\}:(\vec{\tau}')] & (\alpha) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{y},\vec{v},\vec{x})\ \mathsf{be}\ \vec{v}\otimes M\ \mathsf{in}\ N:(\vec{\tau}')] & (\sigma_{\mathrm{var}}) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ (\mathsf{letrec}\ (\vec{v},\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{v})\ \mathsf{in}\ N:(\vec{\tau}')] & (\mathrm{ass}_1) \\
= & [\Gamma\vdash \mathsf{letrec}\ (\vec{v},\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{v}:(\vec{\tau})];[\vec{y}:\vec{\tau}\vdash N:(\vec{\tau}')] & \\
= & Tr^{(\vec{\sigma})}([\Gamma,\vec{x}:\vec{\sigma}\vdash M:(\vec{\tau},\vec{\sigma})]);[\vec{y}:\vec{\tau}\vdash N:(\vec{\tau}')] &
\end{array}$$

- Sliding:

$$
\begin{array}{rll}
& Tr^{(\vec{\sigma})}([\vec{y}:\vec{\tau},\vec{x}:\vec{\sigma}\vdash M:(\vec{\tau}',\vec{\sigma'})];(id_{(\vec{\tau}')}\otimes[\vec{z}:\vec{\sigma}'\vdash N:(\vec{\sigma})])) & \\
= & Tr^{(\vec{\sigma})}([\vec{y}:\vec{\tau},\vec{x}:\vec{\sigma}\vdash M:(\vec{\tau}',\vec{\sigma'})];[\vec{u}:\vec{\tau}',\vec{z}:\vec{\sigma}'\vdash \vec{u}\otimes N:(\vec{\tau}',\vec{\sigma})]) & \\
= & Tr^{(\vec{\sigma})}([\vec{y}:\vec{\tau},\vec{x}:\vec{\sigma}\vdash \mathsf{letrec}\ (\vec{u},\vec{z})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{u}\otimes N:(\vec{\tau}',\vec{\sigma})]) & \\
= & [\vec{y}:\vec{\tau}\vdash \mathsf{letrec}\ (\vec{v},\vec{x})\ \mathsf{be}\ (\mathsf{letrec}\ (\vec{u},\vec{z})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{u}\otimes N)\ \mathsf{in}\ \vec{v}:(\vec{\tau}')] & \\
= & [\vec{y}:\vec{\tau}\vdash \mathsf{letrec}\ (\vec{v},\vec{x},\vec{u},\vec{z})\ \mathsf{be}\ \vec{u}\otimes N\otimes M\ \mathsf{in}\ \vec{v}:(\vec{\tau}')] & (\mathrm{ass}_1) \\
= & [\vec{y}:\vec{\tau}\vdash \mathsf{letrec}\ (\vec{x},\vec{u},\vec{z})\ \mathsf{be}\ N\otimes M\ \mathsf{in}\ \vec{u}:(\vec{\tau}')] & (\sigma_{\mathrm{var}}) \\
= & [\vec{w}:\vec{\tau}\vdash \mathsf{letrec}\ (\vec{x},\vec{s},\vec{z})\ \mathsf{be}\ N\otimes M\{\vec{w}/\vec{y}\}\ \mathsf{in}\ \vec{s}:(\vec{\tau}')] & (\alpha) \\
= & [\vec{w}:\vec{\tau}\vdash \mathsf{letrec}\ (\vec{y},\vec{x},\vec{s},\vec{z})\ \mathsf{be}\ \vec{w}\otimes N\otimes M\ \mathsf{in}\ \vec{s}:(\vec{\tau}')] & (\sigma_{\mathrm{var}}) \\
= & [\vec{w}:\vec{\tau}\vdash \mathsf{letrec}\ (\vec{s},\vec{z},\vec{y},\vec{x})\ \mathsf{be}\ M\otimes\vec{w}\otimes N\ \mathsf{in}\ \vec{s}:(\vec{\tau}')] & (\mathrm{perm}) \\
= & [\vec{w}:\vec{\tau}\vdash \mathsf{letrec}\ (\vec{s},\vec{z})\ \mathsf{be}\ (\mathsf{letrec}\ (\vec{y},\vec{x})\ \mathsf{be}\ \vec{w}\otimes N\ \mathsf{in}\ M)\ \mathsf{in}\ \vec{s}:(\vec{\tau}')] & (\mathrm{ass}_1) \\
= & Tr^{(\vec{\sigma}')}([\vec{w}:\vec{\tau},\vec{z}:\vec{\sigma}'\vdash \mathsf{letrec}\ (\vec{y},\vec{x})\ \mathsf{be}\ \vec{w}\otimes N\ \mathsf{in}\ M:(\vec{\tau}',\vec{\sigma'})]) & \\
= & Tr^{(\vec{\sigma}')}([\vec{w}:\vec{\tau},\vec{z}:\vec{\sigma}'\vdash \vec{w}\otimes N:(\vec{\tau},\vec{\sigma})];[\vec{y}:\vec{\tau},\vec{x}:\vec{\sigma}\vdash M:(\vec{\tau}',\vec{\sigma'})]) & \\
= & Tr^{(\vec{\sigma}')}((id_{(\vec{\tau})}\otimes[\vec{z}:\vec{\sigma}'\vdash N:(\vec{\sigma})]);[\vec{y}:\vec{\tau},\vec{x}:\vec{\sigma}\vdash M:(\vec{\tau}',\vec{\sigma'})]) &
\end{array}
$$

□

Theorem 6.3.2 (completeness)
Given a theory $\mathbb{T}$, there is a complete model in $\mathcal{F}_{\mathbb{T}}:\mathcal{C}_{\mathbb{T}}\to\mathcal{S}_{\mathbb{T}}$, given by $[\![\sigma]\!]=\sigma$ and $[\![F]\!]=[\vec{x}:\vec{\sigma}\vdash F(\vec{x}):(\vec{\tau})]$ for $F:(\vec{\sigma})\to(\vec{\tau})$. □

Theorem 6.3.3

$$\mathbf{CcTrSMC}((\mathcal{C}_{\mathbb{T}}\xrightarrow{\mathcal{F}_{\mathbb{T}}}\mathcal{S}_{\mathbb{T}}),(\mathcal{C}\xrightarrow{\mathcal{F}}\mathcal{S}))\simeq\mathbf{CSharingMod}(\mathbb{T},(\mathcal{C}\xrightarrow{\mathcal{F}}\mathcal{S})).$$

□

Chapter 7

Recursion from Cyclic Sharing

We have studied models of both higher-order acyclic sharing theories (Chapter 4) and (first-order) cyclic sharing theories (Chapter 6), as two orthogonal extensions of (first-order) acyclic sharing theories (Chapter 3). Now we are ready to deal with the most interesting setting in this thesis: the combination of the higher-order extension with cyclic sharing. The consequence is more than an ad hoc combination of two ideas. First, we get a wider class of models of recursive computation, which properly contains the traditional models such as domain theoretic examples but also some non-traditional examples, which are suitable for explaining recursive computation from resource-sensitive cyclic computation, i.e. *recursion from cyclic sharing*. Second, our theory is closely related to *cyclic lambda calculi* (lambda calculi with cyclic sharing) which have been studied as a foundation for practical implementation of functional recursive computation. Therefore our result relates a rich class of models and practically interesting calculi: cyclic lambda calculi serve as languages for our new class of models of recursion, while our models serve as a semantic counterpart of cyclic lambda calculi.

Unlike the previous chapters, we begin with semantic observations, rather than starting from the syntax (which is more or less derivable from the previous ones anyway). We first show a surprisingly simple connection between fixed point operators and traces on cartesian categories. This observation itself stays in the traditional contexts without a notion of sharing, but we then extend this result to cartesian centrally closed traced SMC's, thus our models of higher-order cyclic sharing. After that we introduce higher-order cyclic sharing theories, paying some attention to the related systems known as cyclic lambda calculi. As an application, we analyze fixed point operators definable in cyclic lambda calculi using our semantic models.

Some results in this chapter are also reported in [Has97].

7.1 Fixed Points in Traced Cartesian Categories

Compact closed categories whose monoidal product is cartesian are trivial. This is not the case for traced categories. In fact, in [JSV96] it is shown that the category of sets and binary relations with its biproduct as the monoidal product is traced. Actually we find traced cartesian categories interesting in the context of semantics for recursive computation. Here is a theorem to relate the traditional fixed point operators and traces on cartesian categories, proved by Martin Hyland and the author independently:

Theorem 7.1.1 A cartesian category $\mathcal{C}$ is traced if and only if it has a family of functions

$$(-)^{\dagger_{A,X}} : \mathcal{C}(A \times X, X) \longrightarrow \mathcal{C}(A, X)$$

(in below, parameters A, X may be omitted) such that

1. $(-)^\dagger$ is a *parametrized fixed point operator*; for $f : A \times X \longrightarrow X$, $f^\dagger : A \longrightarrow X$ satisfies $f^\dagger = \langle id_A, f^\dagger \rangle ; f$.
2. $(-)^\dagger$ is natural in A; for $f : A \times X \longrightarrow X$ and $g : B \longrightarrow A$, $((g \times id_X); f)^\dagger = g; f^\dagger : B \longrightarrow X$.
3. $(-)^\dagger$ is natural in X; for $f : A \times X \longrightarrow Y$ and $g : Y \longrightarrow X$, $(f; g)^\dagger = ((id_A \times g); f)^\dagger; g : A \longrightarrow X$.
4. $(-)^\dagger$ satisfies *Bekič's lemma*; for $f : A \times X \times Y \longrightarrow X$ and $g : A \times X \times Y \longrightarrow Y$, $\langle f, g \rangle^\dagger = \langle id_A, (\langle id_{A \times X}, g^\dagger \rangle ; f)^\dagger \rangle ; \langle \pi'_{A,X}, g^\dagger \rangle : A \longrightarrow X \times Y$.

Sketch of the proof: (The full calculation is found in Appendix.) From a trace operator Tr, we define a fixed point operator $(-)^\dagger$ by

$$f^\dagger \ = \ Tr^X(f; \Delta_X) \ : \ A \longrightarrow X$$

for $f : A \times X \longrightarrow X$. Conversely, from a fixed point operator $(-)^\dagger$ we define a trace Tr by

$$Tr^X(f) \ = \ \langle id_A, (f; \pi'_{B,X})^\dagger \rangle ; f; \pi_{B,X} \ : \ A \longrightarrow B$$

(equivalently $((id_A \times \pi'_{B,X}); f)^\dagger; \pi_{B,X}$) for $f : A \times X \longrightarrow B \times X$. We note that these constructions are mutually inverse. $\Box$

There are several equivalent formulations of this result. For instance, in the presence of other conditions, we can restrict 3 to the case that g is a symmetry (c.f. Lemma 1.1. of [JSV96]). For another – practically useful – example, Hyland has shown that axioms 1$\sim$4 are equivalent to 2 and

- *(parametrized) dinaturality*: $(\langle \pi_{A,X}, f\rangle; g)^\dagger = \langle id_A, (\langle \pi_{A,Y}, g\rangle; f)^\dagger\rangle; g : A \longrightarrow X$ for $f : A \times X \longrightarrow Y$ and $g : A \times Y \longrightarrow X$

- *diagonal property*: $(f^\dagger)^\dagger = ((id_A \times \langle id_X, id_X\rangle); f)^\dagger$ for $f : A \times X \times X \longrightarrow X$.

This axiomatization is the same as that of "Conway cartesian categories" in [BÉ96]. Further variations are: 2,4 with dinaturality; and 1,2,4 with the symmetric form of 4.

Perhaps the simplest example is the opposite of the category of sets and partial functions with coproduct as the monoidal product; the trace is given by a form of *feedback* which maps a partial function $f : X \to A+X$ to $f^\dagger : X \to A$, determined by iterating f until we get an answer in A if it exists. Similar settings are studied in detail in [BÉ93].

An immediate consequence of Theorem 7.1.1 is the close relationship between traces and the least fixed point operators in traditional domain theory.

Example 7.1.2 (the least fixed point operator on domains)
Consider the cartesian closed category **Dom** of Scott domains and continuous functions. The least fixed point operator satisfies conditions 1∼4, thus determines a trace operator given by $Tr^X(f) = \lambda a^A.\pi(f(a, \bigcup^n (\lambda x^X.\pi'(f(a,x)))^n(\bot_X)))$ $: A \to B$ for $f : A \times X \to B \times X$. Since the least fixed point operator is the unique dinatural fixed point operator on **Dom**, the trace above is the unique one on **Dom**. □

The same is true for several cartesian closed categories arising from domain theory. In fact, a systematic account is possible. Simpson [Sim93] has shown that, under a mild condition, in cartesian closed full subcategories of the category of algebraic cpo's, the least fixed point operator is characterized as the unique dinatural fixed point operator. On the other hand, it is easy to see that the least fixed point operators satisfy the conditions of Theorem 7.1.1. Therefore, in many such categories, a trace uniquely exists and is determined by the least fixed point operator. However, we note that there are at least two traces in the category of continuous lattices, an instance which does not satisfy Simpson's condition; this category has two fixed point operators which satisfy our conditions – the least one and the greatest continuous one.

Further justification of our axiomatization of fixed point operators comes from recent work on *axiomatic domain theory* which provides a more abstract and systematic treatment of domains and covers a wider range of models of domain theory than the traditional order-theoretic approach. For this, we assume some working knowledge of this topic as found in [Sim92]. Readers who are not familiar with this topic may skip to next section.

Example 7.1.3 (axiomatic domain theory)
Consider a cartesian closed category $\mathcal{C}$ (category of "predomains") equipped with a commutative monad L (the "lift") such that the Kleisli category $\mathcal{C}_L$ (category of predomains and "partial maps") is *algebraically compact* [Fre91]. This setting provides a canonical fixed point operator (derived from the *fixpoint object* [CP92]) on the category of "domains" (obtained as the Kleisli category of the induced comonad on the Eilenberg-Moore category $\mathcal{C}^L$) which satisfies our axioms – Bekič's lemma is proved from the algebraic compactness of $\mathcal{C}_L$ [Mog97] (this idea is due to Plotkin). Thus the requirement for solving recursive domain equations (algebraic compactness) implies that the resulting category of domains is traced. $\Box$

Regarding these facts, we believe that traces provide a good characterization of fixed point operators in traditional denotational semantics.

We conclude this section by observing an attractive fact which suggests how natural our trace-fixpoint correspondence is. Our correspondence preserves a fundamental concept on fixed point operators called *uniformity*, also known as Plotkin's condition. This is important because fixed point operators are often canonically and uniquely characterized by this property.

Proposition 7.1.4 In a traced cartesian category, the following two conditions are equivalent for any $h : X \longrightarrow Y$.

- (Uniformity of the trace operator) For any f and g,
  if $$\begin{tikzcd} A\times X \arrow[r, "f"] \arrow[d, "A\times h"'] & B\times X \arrow[d, "B\times h"] \\ A\times Y \arrow[r, "g"'] & B\times Y \end{tikzcd}$$ commutes then $Tr^X(f) = Tr^Y(g)$.
- (Uniformity of the fixed point operator) For any f and g,
  if $$\begin{tikzcd} A\times X \arrow[r, "f"] \arrow[d, "A\times h"'] & X \arrow[d, "h"] \\ A\times Y \arrow[r, "g"'] & Y \end{tikzcd}$$ commutes then $f^\dagger ; h = g^\dagger$.

<u>Proof:</u> See Appendix. $\Box$

In the case of domain-theoretic categories, the second condition is equivalent to saying that h is a strict map ($\perp$-preserving map). This fact suggests the possibility of studying the notion of strict maps and uniformity of fixed points in more general settings as in the following section. In particular, the first condition makes sense in any traced monoidal category. We remark that this notion of strict maps seems to be far more flexible (or fragile) than that we have in traditional domain theory; in some traced monoidal categories every arrow is strict (e.g. category of sets and partial functions), whereas in some cases only isomorphisms are strict (e.g. category of sets and relations).

Remark 7.1.5 Here is an interesting unanswered question: when do strict maps (defined with respect to the uniformity condition on fixed point operators or traces as above) form a category? Though in many concrete examples (including all domain theoretic examples) strict maps do compose, our abstract definition is not strong enough to ensure that.

More generally, it is straightforward to define the notion of "uniform transformations" (with respect to a class of "strict maps") between mixed variant functors in a similar manner to dinatural transformations [ML71, SD70]. Unlike dinaturals, uniform transformations do compose, but the tradeoff is the failure of the compositionality of (abstract) strict maps. We are not keen to go further in such a general setting, but this question may be a key to develop a theory of recursive computation based on traced categories – we admit that our understanding on this topic is still premature. $\Box$

7.2 Generalized Fixed Points

Our observation so far says that to have an abstract trace is to have a fixed point operator in the traditional sense, provided the monoidal product is cartesian. However, regarding our motivation to model cyclic sharing, this setting is somewhat restrictive – in a cartesian category (regarded as an algebraic theory) arbitrary substitution is justified, thus there is no non-trivial notion of sharing.

Our next step is then to extend this result to our models of cyclic sharing graphs. It turns out that the combination of the higher-order extension with the cyclic sharing theories is sufficient for supporting a generalization of the result of the last section. The essential point is the use of an adjunction between a cartesian category and a traced monoidal category for creating recursion, which is spelled out below.

Let $\mathcal{F} : \mathcal{C} \longrightarrow \mathcal{S}$ be a identity-on-objects strict symmetric monoidal functor from a cartesian category $\mathcal{C}$ to a traced symmetric monoidal category $\mathcal{S}$, with a right adjoint. That is, $\mathcal{F} : \mathcal{C} \longrightarrow \mathcal{S}$ is a cartesian-center traced SMC with a right adjoint.

Theorem 7.2.1 Given $\mathcal{F} : \mathcal{C} \longrightarrow \mathcal{S}$ as above, there is a family of functions

$$(-)^{\dagger_{A,X}} : \mathcal{S}(A \otimes X, X) \longrightarrow \mathcal{S}(A, X)$$

such that

1. $(-)^\dagger$ is a parametrized dinatural fixed point operator: for $f : A \otimes X \longrightarrow Y$ in $\mathcal{S}$ and $g : A \otimes Y \longrightarrow X$ in $\mathcal{S}$, $((\mathcal{F}(\Delta_A) \otimes id_X); (id_A \otimes f); g)^\dagger = \mathcal{F}(\Delta_A); (id_A \otimes ((\mathcal{F}(\Delta_A) \otimes id_Y); (id_A \otimes g); f)^\dagger); g : A \longrightarrow X$.

2. $(-)^\dagger$ is natural in A in $\mathcal{C}$; for $f : A \otimes X \longrightarrow X$ in $\mathcal{S}$ and $g : B \longrightarrow A$ in $\mathcal{C}$, $((\mathcal{F}(g) \otimes id_X); f)^\dagger = \mathcal{F}(g); f^\dagger : B \longrightarrow X$.

Sketch of the proof: (The full calculation is found in Appendix.) Let us write $U : \mathcal{S} \longrightarrow \mathcal{C}$ for the right adjoint of $\mathcal{F}$, and $\epsilon_X : UX \longrightarrow X$ (in $\mathcal{S}$) for the counit. By definition, we have a natural isomorphism $(-)^* : \mathcal{S}(A, B) \overset{\sim}{\longrightarrow} \mathcal{C}(A, UB)$. We also define $\theta_{A,X} : A \times UX \longrightarrow U(A \otimes X)$ in $\mathcal{C}$ by $\theta_{A,X} = (id_A \otimes \epsilon_X)^*$. Now we define $(-)^\dagger$ by

$$f^\dagger = Tr^{UX}(\mathcal{F}(\theta_{A,X}; Uf; \Delta_{UX})); \epsilon_X : A \longrightarrow X \quad \text{in } \mathcal{S}$$

for $f : A \otimes X \longrightarrow X$ in $\mathcal{S}$. $\Box$

The statement above is slightly stronger than Theorem 3.5 in [Has97]. The first condition immediately implies that $(-)^\dagger$ is a parametrized fixed point operator, in the sense that, for $f : A \otimes X \longrightarrow X$ in $\mathcal{S}$, $f^\dagger : A \longrightarrow X$ satisfies $f^\dagger = \mathcal{F}(\Delta_A); (id_A \otimes f^\dagger); f$.

We note that the first condition is equivalent to saying that $(-)^\dagger$ satisfies $(f; g)^\dagger = (g; f)^\dagger; g : I \to X$ for $f : X \to Y$ and $g : Y \to X$ in $\mathcal{S}//A$ (page 50). Since $\mathcal{F}//A : \mathcal{C}//A \to \mathcal{S}//A$ has the structure in the assumption (by Lemma 6.2.3, also it is routine to see that $\mathcal{F}//A$ has a right adjoint $U//A$ given by $U//A(f) = \theta_X; U(f)$), we have just to show this simpler equation in the relativized setting, and this helps us to simplify the proof significantly.

Observe that an easier construction (c.f. Theorem 7.1.1) $Tr^X(f; \mathcal{F}(\Delta_X)) : A \longrightarrow X$ from $f : A \otimes X \longrightarrow X$ in $\mathcal{S}$ does not work as a fixed point operator – the construction in Theorem 7.2.1 uses the adjunction in a crucial manner.

It is in general impossible to recover a trace operator from a fixed point operator which satisfies the conditions of Theorem 7.2.1; for instance, if $\mathcal{S}$ has a zero object 0 such that $0 \otimes A \simeq 0$ (e.g. **Rel** below), the zero map satisfies these conditions. It is an interesting question to ask if we can strengthen the conditions so that we can recover a trace operator, though such conditions seem very delicate. In particular, in this generalized setting, our fixed point operator may not satisfy the Bekič property nor the diagonal property (see Example 7.2.8 below).

A careful inspection of our construction reveals that we need the trace operator just on objects of the form UX (equivalently $\mathcal{F}(UX)$ as $\mathcal{F}$ is identity-on-objects); actually it is sufficient if the full subcategory of $\mathcal{S}$ whose objects are of the form of $UX_1 \otimes \ldots \otimes UX_n$ is traced. Thus such a fixed point operator exists even in a weaker setting. It would be interesting to see if this fixed point operator determines this sub-trace structure. It would be more interesting to see if there is a good connection between such a fixed point operator and fixed point operators in models of intuitionistic linear logic as studied in [Bra95].

An observation corresponding to Proposition 7.1.4 is

Proposition 7.2.2 In the setting as described above, assume that $h : X \to Y$ in $\mathcal{S}$ satisfies the following condition: for any f and g in $\mathcal{S}$, if

$$\begin{tikzcd} A \otimes UX \arrow[r, "f"] \arrow[d, "A\otimes\mathcal{F}(Uh)"'] & B \otimes UX \arrow[d, "B\otimes\mathcal{F}(Uh)"] \\ A \otimes UY \arrow[r, "g"'] & B \otimes UY \end{tikzcd}$$

commutes, then $Tr^{UX}(f) = Tr^{UY}(g)$. Then, for any f and g, if

$$\begin{tikzcd} A \otimes X \arrow[r, "f"] \arrow[d, "A\otimes h"'] & X \arrow[d, "h"] \\ A \otimes Y \arrow[r, "g"'] & Y \end{tikzcd}$$

commutes then $f^{\dagger}; h = g^{\dagger}$.

<u>Proof:</u> See Appendix. $\Box$

Note that our setting is equivalent to saying that we have a cartesian category $\mathcal{C}$ with a monad $T = U \circ \mathcal{F}$ on it, which has a commutative tensorial strength θ, such that the Kleisli category $\mathcal{S} = \mathcal{C}_T$ is traced. So, as we have already noted, it is possible to say that we are dealing with some *notions of computation* in the sense of Moggi [Mog88, Mog91] with extra structure (trace).

Recall that higher-order acyclic sharing theories have been modeled in terms of cartesian centrally closed SMC's, whereas cyclic sharing theories were modeled by cartesian center traced SMC's. To model higher-order cyclic sharing theories, which will be presented and analyzed in the following sections, we use the combination – cartesian centrally closed traced SMC's. That is,

Definition 7.2.3 (cartesian centrally closed traced SMC)
A *cartesian centrally closed traced symmetric monoidal category* (cartesian centrally closed traced SMCs) is an identity-on-objects strict symmetric monoidal functor $\mathcal{F} : \mathcal{C} \longrightarrow \mathcal{S}$ where $\mathcal{C}$ is a cartesian category and $\mathcal{S}$ a traced symmetric monoidal category, such that for every object X the functor $\mathcal{F}(-) \otimes X : \mathcal{C} \longrightarrow \mathcal{S}$ has a right adjoint $X \Rightarrow (-) : \mathcal{S} \longrightarrow \mathcal{C}$. $\Box$

Remark 7.2.4 In [Has97] a cartesian centrally closed traced SMC is called a *traced computational model* (following Moggi's "computational models"). $\Box$

Given a cartesian centrally closed traced SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$, $\mathcal{F}$ itself has a right adjoint $I \Rightarrow (-)$. Therefore

Proposition 7.2.5 In a cartesian centrally closed traced SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$, there is a fixed point operator as described in Theorem 7.2.1. $\Box$

As in the case of cartesian closed categories, we also have an internalized version of the fixed point operator in cartesian centrally closed traced SMC's:

Corollary 7.2.6 Given a cartesian centrally closed traced SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$, there is a dinatural transformation $\mathtt{fix}_{-} : \mathcal{F}((-) \Rightarrow (-)) \longrightarrow (-)$ whose components lie in $\mathcal{S}$ and satisfy $\mathtt{fix}_X = \mathcal{F}(\Delta_{X \Rightarrow X}); (id_{X \Rightarrow X} \otimes \mathtt{fix}_X); \mathbf{ap}_{X,X} : X \Rightarrow X \to X$.

Proof: Let $\mathtt{fix}_X$ be $\mathbf{ap}^{\dagger}_{X,X}$ and observe that $\mathcal{F}(f^*); \mathtt{fix}_X = f^{\dagger}$ for $f : A \otimes X \to X$ in $\mathcal{S}$. $\Box$

To help with the intuition, we shall give a selection of cartesian centrally closed traced SMC's below. Most of them have already been mentioned in the introduction.

Example 7.2.7 (traced cartesian closed categories)
A traced cartesian closed category is a cartesian centrally closed traced SMC in which the cartesian category part and the traced category part are identical. Examples include many domain-theoretic categories such as Example 7.1.2. $\Box$

Example 7.2.8 (non-deterministic model)
The inclusion from the category **Set** of sets and functions to the category **Rel** of sets and binary relations (with the direct product of sets as the symmetric monoidal product) forms a cartesian centrally closed traced SMC: $\mathbf{Rel}(A\otimes X, B) \simeq \mathbf{Set}(A, \mathbf{Rel}(X, B))$. The trace operator on **Rel**, induced by the compact closed structure of **Rel**, is given as follows: for a relation $R : A \otimes X \longrightarrow B \otimes X$, we define a relation $Tr^X(R) : A \longrightarrow B$ by $(a, b) \in Tr^X(R)$ iff $((a, x), (b, x)) \in R$ for an $x \in X$ (here a relation from A to B is given as a subobject of $A \times B$). The parametrized fixed point operator $(-)^\dagger$ on **Rel** is given by

$$R^\dagger \;=\; \{(a,x) \mid \exists S \subseteq X \; S = \{y \mid \exists z \in S \; ((a,z),y) \in R\} \;\&\; x \in S\} \;:\; A \longrightarrow X$$

for $R : A \otimes X \longrightarrow X$ (and $R^\dagger$ is not the zero map!). Its internal version is given by

$$\mathtt{fix}_X = \{(R,x) \mid \exists S \subseteq X \; S; R = S \;\&\; x \in S\} \;:\; \mathbf{Rel}(X,X) \longrightarrow X.$$

This fixed point operator does *not* satisfy the diagonal property (hence Bekič property). For instance, consider $R : \mathrm{bool} \otimes \mathrm{bool} \to \mathrm{bool}$ (where $\mathrm{bool} = \{t, f\}$) such that $(t, f)\ R\ f$ and $(f, t)\ R\ t$. Then $R^\dagger : \mathrm{bool} \to \mathrm{bool}$ is determined by $t\ R^\dagger\ f$ and $f\ R^\dagger\ t$, therefore $R^{\dagger\dagger} \subseteq \mathrm{bool}$ by $\{t, f\}$. However $(\Delta_{\mathrm{bool}}; R)^\dagger = \emptyset$. $\Box$

Note that we can use an elementary topos instead of **Set**, which may provide a computationally more sophisticated model.

An example with more classical flavour:

Example 7.2.9 (finite dimensional vector spaces over a finite field)
Let F_2 be the field with just two elements (thus its characteristic is 2), and $\mathbf{Vect}^{\mathrm{fin}}_{F_2}$ be the category of finite dimensional vector spaces (with chosen bases) over F_2. There is an identity-on-objects strict symmetric monoidal functor from the category of finite sets to $\mathbf{Vect}^{\mathrm{fin}}_{F_2}$ which maps a set S to a vector space with the basis S, and this functor has a right adjoint (the underlying functor). Since $\mathbf{Vect}^{\mathrm{fin}}_{F_2}$ is traced (in the very classical sense), this is an instance of a cartesian centrally closed traced SMC. Note that this example is similar to the previous one – compare the matrix representation of binary relations and that of linear maps. $\Box$

Instead of 2 we can take any other prime number p and use the field F_p of p elements. If the base field is not finite, the identity-on-objects strict functor cannot have a right adjoint, thus fails to be centrally closed.

Example 7.2.10 (higher-order reflexive action calculi)
In Chapter 8 we observe that the higher-order reflexive extension of an action calculus [Mil96, Mil94a, Mil94b] forms a cartesian centrally closed traced SMC. In this calculus the fixed point operator $(-)^\dagger$ is given by

$$a^\dagger \;=\; \uparrow_{\epsilon\Rightarrow n} ((x^{\epsilon\Rightarrow n})^\ulcorner(\mathbf{id}_m \otimes \langle x\rangle \cdot \mathbf{ap}_{\epsilon,n}) \cdot a^\urcorner \cdot \mathbf{copy}_{\epsilon\Rightarrow n}) \cdot \mathbf{ap}_{\epsilon,n} \;:\; m \longrightarrow n$$

for $a : m \otimes n \longrightarrow n$. Mifsud gives essentially the same operator ITER(a) in his thesis [Mif96]. Using this, we can present recursion operators in various process calculi, typically the replication operator. This issue will be further investigated in Chapter 8 (Example 8.3.8). □

7.3 Higher-Order Cyclic Sharing Theory

Now it is almost routine to introduce higher-order cyclic sharing theories – they are obtained by combining cyclic sharing theories and higher-order ones.

Definition 7.3.1 (raw expressions)

$$M \;::=\; x \mid F(M) \mid 0 \mid M_1 \otimes M_2 \mid \mathsf{letrec}\ (x_1,\dots,x_m)\ \mathsf{be}\ M_1\ \mathsf{in}\ M_2 \mid \lambda x.M \mid M_1 M_2$$

Definition 7.3.2 (values)

$$V \;::=\; 0 \mid x \mid \lambda(\vec{x}).M \mid V_1 \otimes V_2$$

□

Definition 7.3.3 (typing)

$$\frac{}{\Gamma, x:\sigma \vdash x : (\sigma)}\ \text{variable}$$

$$\frac{\Gamma \vdash M : (\sigma_1,\dots,\sigma_m) \quad F : (\sigma_1,\dots,\sigma_m) \to (\tau_1,\dots,\tau_n)}{\Gamma \vdash F(M) : (\tau_1,\dots,\tau_n)}\ \text{operator}$$

$$\frac{}{\Gamma \vdash 0 : ()}\ \text{unit}$$

$$\frac{\Gamma \vdash M : (\sigma_1,\dots,\sigma_m) \quad \Gamma \vdash N : (\tau_1,\dots,\tau_n)}{\Gamma \vdash M \otimes N : (\sigma_1,\dots,\sigma_m,\tau_1,\dots,\tau_n)}\ \text{tensor}$$

$$\frac{\begin{array}{c}\Gamma, x_1:\sigma_1,\dots,x_m:\sigma_m \vdash M : (\sigma_1,\dots,\sigma_m)\\ \Gamma, x_1:\sigma_1,\dots,x_m:\sigma_m \vdash N : (\tau_1,\dots,\tau_n)\end{array}}{\Gamma \vdash \mathsf{letrec}\ (x_1,\dots,x_m)\ \mathsf{be}\ M\ \mathsf{in}\ N : (\tau_1,\dots,\tau_n)}\ \text{letrec}$$

$$\frac{\Gamma, x:\sigma, x':\sigma', \Gamma' \vdash M : (\tau_1,\dots,\tau_n)}{\Gamma, x':\sigma', x:\sigma, \Gamma' \vdash M : (\tau_1,\dots,\tau_n)}\ \text{exchange}$$

$$\frac{\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})}{\Gamma \vdash \lambda(\vec{x}).M : ((\vec{\sigma}) \Rightarrow (\vec{\tau}))} \text{ abstraction}$$

$$\frac{\Gamma \vdash M : ((\vec{\sigma}) \Rightarrow (\vec{\tau})) \quad \Gamma \vdash N : (\vec{\sigma})}{\Gamma \vdash MN : (\vec{\tau})} \text{ application}$$

□

Definition 7.3.4 (axioms)

(id)	$\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ \vec{x}$	$= M \quad (\vec{x} \notin FV(M))$
(ass$_1$)	$\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ (\mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ L\ \mathsf{in}\ M)\ \mathsf{in}\ N$	$= \mathsf{letrec}\ (\vec{x}, \vec{y})\ \mathsf{be}\ M \otimes L\ \mathsf{in}\ N$
(ass$_2$)	$\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N$	$= \mathsf{letrec}\ (\vec{x}, \vec{y})\ \mathsf{be}\ L \otimes M\ \mathsf{in}\ N$
($\otimes_1$)	$L \otimes (\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N)$	$= \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ L \otimes N$
($\otimes_2$)	$(\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M) \otimes N$	$= \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M \otimes N$
(perm)	$\mathsf{letrec}\ (\vec{x}, \vec{y}, \vec{z})\ \mathsf{be}\ M_1 \otimes M_2 \otimes M_3\ \mathsf{in}\ N$	$= \mathsf{letrec}\ (\vec{y}, \vec{x}, \vec{z})\ \mathsf{be}\ M_2 \otimes M_1 \otimes M_3\ \mathsf{in}\ N$
(subst)	$\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ F(N)$	$= F(\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N)$
(β)	$(\lambda(\vec{x}).M)N$	$= \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ N\ \mathsf{in}\ M$
(η_0)	$\lambda(\vec{x}).y(\vec{x})$	$= y$
(app$_1$)	$(\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ M)N$	$= \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ L\ \mathsf{in}\ MN$
(app$_2$)	$L(\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N)$	$= \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ LN$
(deref)	$\mathsf{letrec}\ (x, \vec{y})\ \mathsf{be}\ V \otimes M\ \mathsf{in}\ N$	$= \mathsf{letrec}\ (x, \vec{y})\ \mathsf{be}\ V \otimes (M\{V/x\})\ \mathsf{in}\ N\{V/x\}$
(g.c.)	$\mathsf{letrec}\ (x)\ \mathsf{be}\ V\ \mathsf{in}\ N$	$= N \quad (x \notin FV(N) \cup FV(V))$

Both sides of axioms must have the same type under the same context. □

Definition 7.3.5 (higher-order cyclic sharing theory)
A *higher-order cyclic sharing theory* over Σ is an equational theory on the well-typed terms closed under the term constructions described above, where the equality on terms is a congruence relation containing the axioms above. By the *pure higher-order cyclic sharing theory*, we mean the higher-order cyclic sharing theory with no additional axioms. □

As in the first-order case, for readability, we introduce the following syntax for the multiple letrec-binding:

$$\frac{\begin{array}{c}\Gamma, \vec{x}_1 : \vec{\sigma}_1, \ldots, \vec{x}_k : \vec{\sigma}_k \vdash M_i : (\vec{\sigma}_i) \quad (1 \leq i \leq k)\\ \Gamma, \vec{x}_1 : \vec{\sigma}_1, \ldots, \vec{x}_k : \vec{\sigma}_k \vdash N : (\vec{\tau})\end{array}}{\Gamma \vdash \mathsf{letrec}\ (\vec{x}_1)\ \mathsf{be}\ M_1, \ldots\ (\vec{x}_k)\ \mathsf{be}\ M_k\ \mathsf{in}\ N : (\vec{\tau})}$$

for $\mathsf{letrec}\ (\vec{x}_1, \ldots, \vec{x}_k)\ \mathsf{be}\ M_1 \otimes \ldots \otimes M_k\ \mathsf{in}\ N$.

Remark 7.3.6 All axioms except those for substitution of values are inherited from the axioms of cyclic sharing theories and higher-order acyclic sharing theories, which form the first two groups respectively. The axioms for substitution of values, which we name (deref) and (g.c.), come from those for cyclic sharing theories (page 35), and this time we allow the substitutions of values as the axiom (σ_v) of the higher-order acyclic theory. We note that our axioms for substitutions are as strong as "one-step replacement" style substitutions, see below. □

Lemma 7.3.7 The below are derivable:

$$\begin{array}{lcl} \mathsf{letrec}\ (x,\vec{y})\ \mathsf{be}\ V \otimes M[x]\ \mathsf{in}\ N & = & \mathsf{letrec}\ (x,\vec{y})\ \mathsf{be}\ V \otimes M[V]\ \mathsf{in}\ N \\ \mathsf{letrec}\ (x,\vec{y})\ \mathsf{be}\ V \otimes M\ \mathsf{in}\ N[x] & = & \mathsf{letrec}\ (x,\vec{y})\ \mathsf{be}\ V \otimes M\ \mathsf{in}\ N[V] \end{array}$$

where $[x]$ indicates an occurrence of free x in the expression.

<u>Proof:</u> We demonstrate the first case.

$$\begin{array}{cll} & \mathsf{letrec}\ (x)\ \mathsf{be}\ V,\ (\vec{y})\ \mathsf{be}\ M[x]\ \mathsf{in}\ N & \\ = & \mathsf{letrec}\ (x)\ \mathsf{be}\ (\mathsf{letrec}\ (x')\ \mathsf{be}\ V\ \mathsf{in}\ x'),\ (\vec{y})\ \mathsf{be}\ M[x]\ \mathsf{in}\ N & (\text{id}) \\ = & \mathsf{letrec}\ (x)\ \mathsf{be}\ x',\ (x')\ \mathsf{be}\ V,\ (\vec{y})\ \mathsf{be}\ M[x']\ \mathsf{in}\ N & (\text{ass}_1) \\ = & \mathsf{letrec}\ (x')\ \mathsf{be}\ V,\ (x)\ \mathsf{be}\ x',\ (\vec{y})\ \mathsf{be}\ M[x']\ \mathsf{in}\ N & (\text{perm}) \\ = & \mathsf{letrec}\ (x')\ \mathsf{be}\ V,\ (x)\ \mathsf{be}\ V,\ (\vec{y})\ \mathsf{be}\ M[V]\ \mathsf{in}\ N & (\text{deref}) \\ = & \mathsf{letrec}\ (x)\ \mathsf{be}\ V,\ (\vec{y})\ \mathsf{be}\ M[V]\ \mathsf{in}\ N & (\text{g.c.}) \end{array}$$

□

By the same proofs as in Chapter 4 and 6, we have

Lemma 7.3.8 α-conversions of letrec-bindings and lambda-bindings, and β and η axioms restricted to values are derivable. □

The notions of structures and models are defined in the same manner as in Chapter 3, 4 and 6.

Theorem 7.3.9 (soundness)
Let $[\![-]\!]$ be a higher-order cyclic sharing model of a higher-order cyclic sharing theory. If $\Gamma \vdash M = N : (\vec{\sigma})$ is derivable in the theory, then $[\![\Gamma \vdash M : (\vec{\sigma})]\!] = [\![\Gamma \vdash N : (\vec{\sigma})]\!]$. □

Also we have the classifying category $\mathcal{F}_{\mathbb{T}} : \mathcal{C}_{\mathbb{T}} \to \mathcal{S}_{\mathbb{T}}$ for a higher-order cyclic sharing theory $\mathbb{T}$ and a complete model (generic model) in it.

Theorem 7.3.10 (completeness)
Given a theory $\mathbb{T}$, there is a complete model in $\mathcal{F}_{\mathbb{T}} : \mathcal{C}_{\mathbb{T}} \to \mathcal{S}_{\mathbb{T}}$, given by $[\![\sigma]\!] = \sigma$ and $[\![F]\!] = [\vec{x} : \vec{\sigma} \vdash F(\vec{x}) : (\vec{\tau})]$ for $F : (\vec{\sigma}) \to (\vec{\tau})$. □

Example 7.3.11 The fixed point operator in Theorem 7.2.1 can be represented using the higher-order cyclic sharing theory as an internal language of such categorical structures, as

$$\frac{\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\sigma})}{\Gamma \vdash \mu(\vec{x}).M \;\equiv\; \mathsf{letrec}\ (f)\ \mathsf{be}\ \lambda().(\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ f0\ \mathsf{in}\ M)\ \mathsf{in}\ f0 : (\vec{\sigma})}$$

Let us show dinaturality using the equational theory. Assume that $\Gamma, \vec{x} : \vec{\sigma} \vdash M : (\vec{\tau})$ and $\Gamma, \vec{y} : \vec{\tau} \vdash N : (\vec{\sigma})$. Then we have compositions of them $M \circ N$ and $N \circ M$ by

$$\Gamma, \vec{x} : \vec{\sigma} \vdash M \circ N \;\equiv\; \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ M\ \mathsf{in}\ N : (\vec{\sigma})$$

and

$$\Gamma, \vec{y} : \vec{\tau} \vdash N \circ M \;\equiv\; \mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ N\ \mathsf{in}\ M : (\vec{\tau}).$$

Also we define values V_M and V_N by

$$\Gamma, f : () \Rightarrow (\vec{\sigma}) \vdash V_M \;\equiv\; \lambda().(\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ f0\ \mathsf{in}\ M) : (() \Rightarrow (\vec{\tau}))$$

and

$$\Gamma, g : () \Rightarrow (\vec{\tau}) \vdash V_N \;\equiv\; \lambda().(\mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ g0\ \mathsf{in}\ N) : (() \Rightarrow (\vec{\sigma})).$$

Then we can show

$$\begin{aligned}
\Gamma, f : () \Rightarrow (\vec{\sigma}) \vdash \mathsf{letrec}\ (g)\ \mathsf{be}\ V_M\ \mathsf{in}\ V_N &= \lambda().(\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ f0\ \mathsf{in}\ N \circ M) : (() \Rightarrow (\vec{\sigma}))\\
\Gamma, g : () \Rightarrow (\vec{\tau}) \vdash \mathsf{letrec}\ (f)\ \mathsf{be}\ V_N\ \mathsf{in}\ V_M &= \lambda().(\mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ g0\ \mathsf{in}\ M \circ N) : (() \Rightarrow (\vec{\tau}))
\end{aligned}$$

using β and axioms for substitutions. Now dinaturality is proved as

$$\begin{array}{cll}
& \mu(\vec{x}).(N \circ M) & \\
= & \mathsf{letrec}\ (f)\ \mathsf{be}\ \lambda().(\mathsf{letrec}\ (\vec{x})\ \mathsf{be}\ f0\ \mathsf{in}\ N \circ M)\ \mathsf{in}\ f0 & \\
= & \mathsf{letrec}\ (f)\ \mathsf{be}\ (\mathsf{letrec}\ (g)\ \mathsf{be}\ V_M\ \mathsf{in}\ V_N)\ \mathsf{in}\ f0 & \\
= & \mathsf{letrec}\ (f)\ \mathsf{be}\ V_N,\ (g)\ \mathsf{be}\ V_M\ \mathsf{in}\ f0 & (\mathrm{ass}_1)\\
= & \mathsf{letrec}\ (f)\ \mathsf{be}\ V_N,\ (g)\ \mathsf{be}\ V_M\ \mathsf{in}\ V_N 0 & (\mathrm{deref})\\
= & \mathsf{letrec}\ (f)\ \mathsf{be}\ V_N,\ (g)\ \mathsf{be}\ V_M\ \mathsf{in}\ \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ g0\ \mathsf{in}\ N & (\beta_v)\\
= & \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ (\mathsf{letrec}\ (g)\ \mathsf{be}\ V_M,\ (f)\ \mathsf{be}\ V_N\ \mathsf{in}\ g0)\ \mathsf{in}\ N & (\mathrm{ass}_1)\\
= & \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ (\mathsf{letrec}\ (g)\ \mathsf{be}\ (\mathsf{letrec}\ (f)\ \mathsf{be}\ V_N\ \mathsf{in}\ V_M)\ \mathsf{in}\ g0)\ \mathsf{in}\ N & \\
= & \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ (\mathsf{letrec}\ (g)\ \mathsf{be}\ \lambda().(\mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ g0\ \mathsf{in}\ M \circ N)\ \mathsf{in}\ g0)\ \mathsf{in}\ N & \\
= & \mathsf{letrec}\ (\vec{y})\ \mathsf{be}\ \mu(\vec{y}).(M \circ N)\ \mathsf{in}\ N. &
\end{array}$$

□

7.4 Cyclic Lambda Calculi

As a fragment and a variant of higher-order cyclic sharing theories, we introduce two simply typed lambda calculi enriched with the notion of cyclic sharing, the *simply typed* $\lambda_{\mathrm{letrec}}$*-calculus* and $\lambda^{v}_{\mathrm{letrec}}$*-calculus* in which cyclically shared resources are represented in terms of the letrec syntax. Traced cartesian closed categories and cartesian centrally traced SMCs serve as sound and complete models of these calculi respectively.

Basically we deal with a simplified pure higher-order cyclic sharing theory with no multiple conclusion and no operator symbols, thus almost all observations here can be seen as consequences of the results obtained so far. However, since cyclic lambda calculi have their own right to be studied, especially in the connection with graph rewriting theory and implementations of functional programming languages, we spell out them here.

The Syntax and Axioms

We design the simply typed $\lambda^{v}_{\mathrm{letrec}}$-calculus as a modification of (the commutative version of) Moggi's computational lambda calculus (Definition 5.3.4) [Mog88]; we replace the let-syntax by the letrec-syntax which allows cyclic bindings.

In this section, we fix a set of *base types*.

Types

$$\sigma, \tau \ldots \ ::= \ b \mid \sigma \Rightarrow \tau \quad \text{(where } b \text{ is a base type)}$$

Syntactic Domains

Variables	$x, y, z \ldots$
Raw Terms	$M, N \ldots \ ::= \ x \mid \lambda x.M \mid MN \mid \mathsf{letrec}\ D\ \mathsf{in}\ N$
Values	$V, W \ldots \ ::= \ x \mid \lambda x.M$
Declarations	$D \ldots \ ::= \ x = M \mid x = M, D$

In a declaration, binding variables are assumed to be disjoint.

Typing

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ \text{Variable} \qquad \frac{\Gamma, x : \sigma, y : \sigma', \Gamma' \vdash M : \tau}{\Gamma, y : \sigma', x : \sigma, \Gamma' \vdash M : \tau}\ \text{Exchange}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \Rightarrow \tau}\ \text{Abstraction} \qquad \frac{\Gamma \vdash M : \sigma \Rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}\ \text{Application}$$

$$\frac{\Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M_i : \sigma_i \ (i = 1, \ldots, n) \quad \Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash N : \tau}{\Gamma \vdash \mathsf{letrec}\ x_1 = M_1, \ldots, x_n = M_n\ \mathsf{in}\ N : \tau}\ \mathsf{letrec}$$

Axioms

Identity	$\mathsf{letrec}\ x = M\ \mathsf{in}\ x$	$=$	$M \quad (x \notin FV(M))$
Associativity	$\mathsf{letrec}\ y = (\mathsf{letrec}\ D_1\ \mathsf{in}\ M), D_2\ \mathsf{in}\ N$	$=$	$\mathsf{letrec}\ D_1, y = M, D_2\ \mathsf{in}\ N$
	$\mathsf{letrec}\ D_1\ \mathsf{in}\ \mathsf{letrec}\ D_2\ \mathsf{in}\ M$	$=$	$\mathsf{letrec}\ D_1, D_2\ \mathsf{in}\ M$
Permutation	$\mathsf{letrec}\ D_1, D_2, D\ \mathsf{in}\ N$	$=$	$\mathsf{letrec}\ D_2, D_1, D\ \mathsf{in}\ N$
Commutativity	$(\mathsf{letrec}\ D\ \mathsf{in}\ M)N$	$=$	$\mathsf{letrec}\ D\ \mathsf{in}\ MN$
	$M(\mathsf{letrec}\ D\ \mathsf{in}\ N)$	$=$	$\mathsf{letrec}\ D\ \mathsf{in}\ MN$
β	$(\lambda x.M)N$	$=$	$\mathsf{letrec}\ x = N\ \mathsf{in}\ M$
σ_v	$\mathsf{letrec}\ x = V, D[x]\ \mathsf{in}\ M$	$=$	$\mathsf{letrec}\ x = V, D[V]\ \mathsf{in} M$
	$\mathsf{letrec}\ x = V, D\ \mathsf{in}\ M[x]$	$=$	$\mathsf{letrec}\ x = V, D\ \mathsf{in} M[V]$
	$\mathsf{letrec}\ x = V\ \mathsf{in}\ M$	$=$	$M\ (x \notin FV(V) \cup FV(M))$
η_0	$\lambda x.yx$	$=$	y

Both sides of equations must have the same type under the same typing context; we will work just on well-typed terms. We assume the usual conventions on variables.

We remark that axioms Identity, Associativity, Permutation and Commutativity ensure that two $\lambda^v_{\mathrm{letrec}}$-terms are identified if they correspond to the same cyclic directed graph; thus they are a sort of structural congruence, rather than representing actual computation. β creates a sharing from a function application. σ_v describes the substitution of values (the first two for the dereference, the last one for the garbage collection). In $M[x]$ and $D[x]$, $[x]$ denotes a free occurrence of x. From β, σ_v and η_0, we have the "call-by-value" $\beta\eta$-equations:

Lemma 7.4.1 In $\lambda^v_{\mathrm{letrec}}$-calculus, the following are derivable.

$$\begin{array}{llcll} \beta_v & (\lambda x.M)V & = & M\{V/x\} & \\ \eta_v & (\lambda x.Vx) & = & V \quad (x \notin FV(V)) & \square \end{array}$$

We think it is misleading to relate this calculus to the call-by-value operational semantics; restricting substitutions on values does not mean that this calculus is call-by-value. Rather, our equational theory is fairly close to the *call-by-need* calculus proposed in [AF96], which corresponds to a version of lazy implementations of the call-by-name operational semantics. We expect that this connection is the right direction to relate our calculus to an operational semantics.

Also we define a "strengthened" version in which arbitrary substitution and η-reduction are allowed (thus any term is a value):

$$\begin{array}{llcl} \sigma & \mathsf{letrec}\ x = N, D[x]\ \mathsf{in}\ M & = & \mathsf{letrec}\ x = N, D[N]\ \mathsf{in} M \\ & \mathsf{letrec}\ x = N, D\ \mathsf{in}\ M[x] & = & \mathsf{letrec}\ x = N, D\ \mathsf{in} M[N] \\ & \mathsf{letrec}\ x = N\ \mathsf{in}\ M & = & M \quad (x \notin FV(M)) \\ \eta & \lambda x.Mx & = & M \quad (x \notin FV(M)) \end{array}$$

We shall call this version the *simply typed* $\lambda_{\mathrm{letrec}}$*-calculus* – this corresponds to the calculus in [AK94] ignoring the typing and the extensionality (η-axiom).

Interpretation into Models

We spell out how to interpret our cyclic lambda calculi in traced cartesian closed categories as well as cartesian centrally closed traced SMC's. Again it is essentially a simplification of the previous interpretations of sharing theories.

We just present the case of the $\lambda^v_{\text{letrec}}$-calculus; the case of the λ_{letrec}-calculus is obtained just by replacing a cartesian centrally closed traced SMC by a traced cartesian closed category.

Let us fix a cartesian centrally closed traced SMC $\mathcal{F} : \mathcal{C} \longrightarrow \mathcal{S}$, and choose an object $[\![b]\!]$ for each base type b. The interpretation of arrow types is then defined by $[\![\sigma \Rightarrow \tau]\!] = [\![\sigma]\!] \Rightarrow [\![\tau]\!]$. We interpret a $\lambda^v_{\text{letrec}}$-term (with its typing environment) $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \tau$ by an arrow $[\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \tau]\!] : [\![\sigma_1]\!] \otimes \ldots \otimes [\![\sigma_n]\!] \longrightarrow [\![\tau]\!]$ in $\mathcal{S}$ as follows.

$$\begin{array}{lcl}
[\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash x_i : \sigma_i]\!] & = & \mathcal{F}\pi_i \;\text{ where } \pi_i \text{ is the } i\text{-th projection} \\
[\![\Gamma \vdash \lambda x.M : \sigma \Rightarrow \tau]\!] & = & \mathcal{F}(([\![\Gamma, x : \sigma \vdash M : \tau]\!])^*) \\
[\![\Gamma \vdash M^{\sigma \Rightarrow \tau} N^{\sigma} : \tau]\!] & = & \mathcal{F}\Delta; ([\![\Gamma \vdash M : \sigma \Rightarrow \tau]\!] \otimes [\![\Gamma \vdash N : \tau]\!]); \mathbf{ap} \\
[\![\Gamma \vdash \mathsf{letrec}\ x_1 = M_1^{\sigma_1}, .., x_k = M_k^{\sigma_k} \ \mathsf{in}\ N : \tau]\!] \;= & & \\
\multicolumn{3}{l}{\mathcal{F}\Delta; (id \otimes Tr^{[\![\sigma_1]\!] \otimes .. [\![\sigma_k]\!]}(\mathcal{F}\Delta_k; ([\![\Gamma' \vdash M_1 : \sigma_1]\!] \otimes \ldots [\![\Gamma' \vdash M_k : \sigma_k]\!]); \mathcal{F}\Delta)); [\![\Gamma' \vdash N : \tau]\!]}
\end{array}$$

Recall that $\mathbf{ap}_{A,B} : (A \Rightarrow B) \otimes A \longrightarrow B$ is the counit of the adjoint $\mathcal{F}(-) \otimes A \dashv A \Rightarrow (-)$, and $(-)^* : \mathcal{T}(\mathcal{F}A \otimes B, C) \longrightarrow \mathcal{C}(A, B \Rightarrow C)$ is the associated natural bijection. In the last case, Γ' is $\Gamma, x_1 : \sigma_1, \ldots, x_k : \sigma_k$ and Δ_{kA} is the k-times copy from A to $\underbrace{A \times \ldots \times A}_{k \text{ times}}$. Note that values are first interpreted in $\mathcal{C}$ (following Moggi's account, $\mathcal{C}$ is the category of values) and then lifted to $\mathcal{S}$ via $\mathcal{F}$.

Calculations similar to those in previous chapters show that cartesian centrally closed traced SMCs are sound for the $\lambda^v_{\text{letrec}}$-calculus (and the same for traced cartesian closed categories and the λ_{letrec}-calculus):

Theorem 7.4.2 (Soundness)

- For any cartesian centrally closed traced SMC with chosen object $[\![b]\!]$ for each base type b, this interpretation is sound; if $\Gamma \vdash M : \sigma$, $\Gamma \vdash N : \sigma$ and $M = N$ in the $\lambda^v_{\text{letrec}}$-calculus then $[\![\Gamma \vdash M : \sigma]\!] = [\![\Gamma \vdash N : \sigma]\!]$.

- For any traced cartesian closed category with chosen object $[\![b]\!]$ for each base type b, this interpretation is sound; if $\Gamma \vdash M : \sigma$, $\Gamma \vdash N : \sigma$ and $M = N$ in the λ_{letrec}-calculus then $[\![\Gamma \vdash M : \sigma]\!] = [\![\Gamma \vdash N : \sigma]\!]$. □

Example 7.4.3 (domain-theoretic model)
As we already noted, **Dom** is a traced cartesian closed category (hence also a cartesian centrally closed traced SMC). The interpretation of a λ_{letrec}-term $\vdash \mathsf{letrec}\ x = M\ \mathsf{in}\ x : \sigma$ in **Dom** is just the least fixed point $\bigcup_n F^n(\bot)$ where $F : [\![\sigma]\!] \longrightarrow [\![\sigma]\!]$ is the interpretation of $x : \sigma \vdash M : \sigma$. $\Box$

Similarly, the traced cartesian closed category of ω-cpo's with bottoms serves as a sound model of the λ_{letrec}-calculus. Since there is a faithful interpretation of the simply typed lambda calculus (with no constant) in this category (due to Plotkin, c.f. Theorem 2 of [Sim95]) which factors through the interpretation of the λ_{letrec}-calculus, we have

Corollary 7.4.4 The λ_{letrec}-calculus is a conservative extension of the simply typed lambda calculus. $\Box$

which can be stated more semantically as

Proposition 7.4.5 The free cartesian closed category generated by a set of objects faithfully embeds into the free traced cartesian closed category (with the same objects). $\Box$

Remark 7.4.6 For non-free cases, adding a trace while preserving the cartesian (closed) structure may cause a collapse. Plotkin has shown that there is no finite product preserving embedding from a cartesian category with a Mal'cev operator into a traced cartesian category, by appealing to the observation by Plotkin and Simpson (Theorem 10 in [Sel96]) that algebraic theories with a Mal'cev operator and a fixed point operator are inconsistent. $\Box$

Example 7.4.7 (non-deterministic model)
In **Rel** (Example 7.2.8), a $\lambda^v_{\text{letrec}}$-term is interpreted as the set of "all possible solutions of the recursive equation". The interpretation of $\vdash \mathsf{letrec}\ x = M\ \mathsf{in}\ x : \sigma$ is just the set $\{x \in [\![\sigma]\!] \mid (x,x) \in [\![x : \sigma \vdash M : \sigma]\!]\}$ (a subobject of $[\![\sigma]\!] = 1 \times [\![\sigma]\!]$). For instance,

$$\begin{array}{lcll}
[\![\vdash \mathsf{letrec}\ x = x\ \mathsf{in}\ x : \sigma]\!] & = & [\![\sigma]\!] & : 1 \longrightarrow [\![\sigma]\!] \\
[\![\vdash \mathsf{letrec}\ x = x^2\ \mathsf{in}\ x : \mathrm{nat}]\!] & = & \{0,1\} & : 1 \longrightarrow \mathbf{N} \\
[\![\vdash \mathsf{letrec}\ x = x+1\ \mathsf{in}\ x : \mathrm{nat}]\!] & = & \emptyset & : 1 \longrightarrow \mathbf{N}
\end{array}$$

(for the latter two cases we enrich the calculus with natural numbers). Note that this model is sound for the $\lambda^v_{\text{letrec}}$-calculus, but not for the λ_{letrec}-calculus – since we cannot copy non-deterministic computation, this model is "resource-sensitive". $\Box$

This non-deterministic model is not a complete (faithful) model of the $\lambda^v_{\mathrm{letrec}}$-calculus. However, we conjecture that the commutative computational lambda calculus faithfully embeds into this model, which implies that the $\lambda^v_{\mathrm{letrec}}$-calculus is conservative over the commutative computational lambda calculus.

The classifying category of the pure higher-order sharing theory serves as a complete model of the $\lambda^v_{\mathrm{letrec}}$-calculus. Thus we also have completeness:

Theorem 7.4.8 (Completeness)

- If $[\![\Gamma \vdash M : \sigma]\!] = [\![\Gamma \vdash N : \sigma]\!]$ for every cartesian centrally closed traced SMC, then $M = N$ in the $\lambda^v_{\mathrm{letrec}}$-calculus.
- If $[\![\Gamma \vdash M : \sigma]\!] = [\![\Gamma \vdash N : \sigma]\!]$ for every traced cartesian closed category, then $M = N$ in the $\lambda_{\mathrm{letrec}}$-calculus. $\Box$

Remark 7.4.9 To represent the parametrized fixed point operator given in Theorem 7.2.1 we have to extend the $\lambda^v_{\mathrm{letrec}}$-calculus with a *unit type* unit which has a unique value $*$:

$$\frac{}{\Gamma \vdash * : \mathsf{unit}}\,\mathrm{Unit} \qquad V \;=\; * \quad (V : \mathsf{unit})$$

The interpretation of the unit type in a cartesian centrally closed traced SMC is just the terminal object (unit object). The type constructor $\mathsf{unit} \Rightarrow (-)$ then plays the role of the right adjoint of the inclusion from the category of values to the category of terms. We define the parametrized fixed point operator by

$$\frac{\Gamma, x : \sigma \vdash M : \sigma}{\Gamma \vdash \mu x^\sigma . M \;\equiv\; \mathsf{letrec}\ f^{\mathsf{unit} \Rightarrow \sigma} = \lambda y^{\mathsf{unit}}.((\lambda x^\sigma . M)(f *))\ \mathsf{in}\ f * \;:\; \sigma}$$

which satisfies $\mu x.M = (\lambda x.M)(\mu x.M)$, but may not satisfy the standard fixed point equation $\mu x.M = M\{\mu x.M/x\}$ in the $\lambda^v_{\mathrm{letrec}}$-calculus because $\mu x.M$ may not be a value in general. The operator Y_3 in the next section is essentially same as this fixed point operator, except for avoiding to use unit. $\Box$

We could give the untyped version and its semantic models – by a reflexive object in a cartesian centrally closed traced SMC (or a traced cartesian closed category). Regarding the results in previous sections, we can establish the connection between the dinatural diagonal fixed point operator in a model of the untyped $\lambda_{\mathrm{letrec}}$-calculus and the trace operator of the cartesian closed category. It would be interesting to compare recursion created by untypedness and recursion created by trace (cyclic sharing) in such models.

Comparison with Ariola-Blom-Klop Approach

A detailed study of the equational and rewriting-theoretic aspects of cyclic lambda calculi has been done by Ariola and Blom [AB97], following the previous work by Ariola and Klop [AK94]. Our cyclic lambda calculi fit into their account, but a few remarks should follow. Our equational theories (for the $\lambda_{\mathrm{letrec}}$- and $\lambda^{v}_{\mathrm{letrec}}$-calculi) are strictly weaker than the corresponding equational theories in [AB97] ignoring the eta axiom η_0. This is because

- Our equational theories correspond to what Ariola and Blom call "scoped lambda graphs" where two lambda graphs are separated if they have different scoping of variable bindings even when they have the same underlying graph. Ariola and Blom introduce axioms for identifying them.
- Ariola and Blom introduce more axioms to equate graphs which are bisimular, i.e. have the same (infinite) tree unwinding.
- For the $\lambda^{v}_{\mathrm{letrec}}$-calculus, the notion of values is slightly different, in that Ariola and Blom treat cyclic terms which contain only values in a similar manner to values.

Assuming η_0, their equational theory is a quotient of ours. Ariola and Blom have shown that their calculus is sound and complete for their infinite tree unwinding semantics (this is a highly non-trivial result as syntactically cyclic lambda calculi have no confluent rewriting system [AK94], therefore no easy term model exists; Ariola and Blom overcome this problem by showing a semantic confluence result up to the contents of "information" in terms). We conjecture that their semantic models give rise to a traced cartesian closed category and a cartesian centrally closed traced SMC.

Therefore, equationally our (pure) theories are essentially proper subsets of the "most complete" calculi in [AB97]. However, interestingly, as rewriting systems, ours are as strong as theirs; Ariola and Blom have shown in [AB97] that, regarded as rewriting systems, our calculi are complete for their infinite semantics. It is interesting to see if the infinite semantics, conjectured to form instances of our models, stand out as models with some "good" characterization – at least we should like to know the theoretical justification of why infinite trees models have occupied an important position as the canonical semantic models of rewriting systems for long time.

7.5 Analyzing Fixed Points

In the $\lambda^v_{\text{letrec}}$-calculus, several (weak) fixed point operators are definable – this is not surprising, because there are several known encodings of fixed point operators in terms of cyclic sharing. However, it is difficult to see that they are not identified by our equational theory – syntactic reasoning for cyclic graph structures is not an easy task, as the non-confluency result in [AK94] suggests. On the other hand, in many traditional models for recursive computation, all of them have the same denotational meaning mainly because we cannot distinguish values from non-values in such models.

One purpose in developing the models of higher-order cyclic sharing theories is to give a clear semantic account for these several recursive computations created from cyclic sharing. Though this topic has not yet been fully developed, we shall give some elementary analysis using the $\lambda^v_{\text{letrec}}$-calculus and a model (**Rel**).

We define $\lambda^v_{\text{letrec}}$-terms $\Gamma \vdash Y_i(M) : \sigma$ ($i = 1, 2, 3$) for given term $\Gamma \vdash M : \sigma \Rightarrow \sigma$ as follows.

$$\begin{array}{lcl}
Y_1 & = & \mathsf{letrec}\ \mathrm{fix}^{(\sigma\Rightarrow\sigma)\Rightarrow\sigma} = \lambda f^{\sigma\Rightarrow\sigma}.f(\mathrm{fix}\ f)\ \mathsf{in}\ \mathrm{fix} \\
Y_2 & = & \lambda f^{\sigma\Rightarrow\sigma}.\mathsf{letrec}\ x^\sigma = fx\ \mathsf{in}\ x \\
Y_3(M) & = & \mathsf{letrec}\ g^{\tau\Rightarrow\sigma} = \lambda y^\tau.M(gy)\ \mathsf{in}\ gN \\
 & & (N \text{ is a closed term of type } \tau, \text{ e.g. } \mathsf{letrec}\ x = x\ \mathsf{in}\ x\ : \tau)
\end{array}$$

Each of them can be used as a fixed point operator, but their behaviours are not the same. For instance, it is known that Y_2 is more efficient than others, under the call-by-need evaluation strategy [Lau93]. Y_1 satisfies the fixed point equation $YV = V(YV)$ for any value $V : \sigma \Rightarrow \sigma$.

$$\begin{array}{rcll}
Y_1 M & = & \mathsf{letrec}\ \mathrm{fix} = \lambda f.f(\mathrm{fix}\ f)\ \mathsf{in}\ \mathrm{fix}M & \text{Commutativity} \\
 & = & \mathsf{letrec}\ \mathrm{fix} = \lambda f.f(\mathrm{fix}\ f)\ \mathsf{in}\ (\lambda f.f(\mathrm{fix}\ f))M & \sigma_v \\
 & = & \mathsf{letrec}\ \mathrm{fix} = \lambda f.f(\mathrm{fix}\ f)\ \mathsf{in}\ \mathsf{letrec}\ f' = M\ \mathsf{in}\ f'(\mathrm{fix}\ f') & \beta \\
 & = & \mathsf{letrec}\ f' = M\ \mathsf{in}\ \mathsf{letrec}\ \mathrm{fix} = \lambda f.f(\mathrm{fix}\ f)\ \mathsf{in}\ f'(\mathrm{fix}\ f') & \text{Assoc., Perm.} \\
 & = & \mathsf{letrec}\ f' = M\ \mathsf{in}\ f'((\mathsf{letrec}\ \mathrm{fix} = \lambda f.f(\mathrm{fix}\ f)\ \mathsf{in}\ \mathrm{fix})f') & \text{Commutativity} \\
 & = & \mathsf{letrec}\ f' = M\ \mathsf{in}\ f'(Y_1 f') & \\
 & (= & M(Y_1 M) \quad \text{if } M \text{ is a value}) &
\end{array}$$

Y_2 satisfies $Y_2 M = M(Y_2 M)$ only when Mx is equal to a value (hence M is supposed to be a higher-order value). If $M = \lambda y.V$ for some value V,

$$\begin{array}{rcll}
Y_2 M & = & \mathsf{letrec}\ x = (\lambda y.V)x\ \mathsf{in}\ x & \beta_v \\
 & = & \mathsf{letrec}\ x = V\{x/y\}\ \mathsf{in}\ x & \beta_v \\
 & = & \mathsf{letrec}\ x = V\{x/y\}\ \mathsf{in}\ V\{x/y\} & \sigma_v \\
 & = & \mathsf{letrec}\ x = (\lambda y.V)x\ \mathsf{in}\ (\lambda y.V)x & \beta_v \\
 & = & (\lambda y.V)(\mathsf{letrec}\ x = (\lambda y.V)x\ \mathsf{in}\ x) & \text{Commutativity} \\
 & = & M(Y_2 M) &
\end{array}$$

Y_3 satisfies $Y_3(M) = M(Y_3(M))$ for any term $M : \sigma \Rightarrow \sigma$ (thus is a "true" fixed point operator).

$$\begin{array}{rcll}
Y_3(M) & = & \mathsf{letrec}\ g = \lambda y.M(gy)\ \mathsf{in}\ gN & \\
& = & \mathsf{letrec}\ g = \lambda y.M(gy)\ \mathsf{in}\ (\lambda y.M(gy))N & \sigma_v \\
& = & \mathsf{letrec}\ g = \lambda y.M(gy)\ \mathsf{in}\ \mathsf{letrec}\ y' = N\ \mathsf{in}\ M(gy') & \beta \\
& = & \mathsf{letrec}\ g = \lambda y.M(gy)\ \mathsf{in}\ M(g(\mathsf{letrec}\ y' = N\ \mathsf{in}\ y')) & \text{Commutativity} \\
& = & M(\mathsf{letrec}\ g = \lambda y.M(gy)\ \mathsf{in}\ g(\mathsf{letrec}\ y' = N\ \mathsf{in}\ y')) & \text{Commutativity} \\
& = & M(\mathsf{letrec}\ g = \lambda y.M(gy)\ \mathsf{in}\ gN) & \text{Identity} \\
& = & M(Y_3(M)) &
\end{array}$$

The interpretation of these operators in a cartesian centrally closed traced SMC is as follows.

$$\begin{array}{lcl}
[\![\vdash Y_1]\!] & = & Tr^{(A\Rightarrow A)\Rightarrow A}(F(\mathbf{cur}((id \otimes \Delta); (\mathbf{ap} \otimes id); c; \mathbf{ap})); \Delta) \\
[\![\vdash Y_2]\!] & = & F(\mathbf{cur}(Tr^{A}(\mathbf{ap}; \Delta))) \\
[\![\Gamma \vdash Y_3(M)]\!] & = & (Tr^{B\Rightarrow A}(F(\mathbf{cur}(([\![\Gamma \vdash M : \sigma \Rightarrow \sigma]\!] \otimes \mathbf{ap}); \mathbf{ap})); \Delta) \otimes [\![\vdash N : \tau]\!]); \mathbf{ap}
\end{array}$$

where $A = [\![\sigma]\!]$ and $B = [\![\tau]\!]$. They have the different interpretations in **Rel**, hence are not identified in the $\lambda^{v}_{\mathrm{letrec}}$-calculus. Assume that $S = [\![\vdash M : \sigma \Rightarrow \sigma]\!] \subseteq$ **Rel**(A, A). Then

$$[\![\vdash Y_1(M) : \sigma]\!] = \bigcup_{f\in S} \bigcup_{(A';f)=A'\subseteq A} A' \qquad [\![\vdash Y_2(M) : \sigma]\!] = \bigcup_{f\in S} \{x \mid (x, x) \in f\}$$

whereas

$$[\![\vdash Y_3(M) : \sigma]\!] = \bigcup_{(A';\bigcup S)=A'\subseteq A} A'$$

(In the definition of Y_3, we take $N : \tau$ as $\mathsf{letrec}\ x = x\ \mathsf{in}\ x : \tau$.)

Remark 7.5.1 If we interpret a $\lambda_{\mathrm{letrec}}$-term of the form $\mathsf{letrec}\ x = M\ \mathsf{in}\ N$ by an untyped term $(\lambda x.N)(Y(\lambda x.M))$ where $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ (Curry's fixed point combinator), the fixed point operators above are related to more or less familiar combinators (c.f. [Bar84], Chapter 6.1):

- Y_1 corresponds to $(\lambda x.\lambda y.y(xxy))(\lambda x.\lambda y.y(xxy))$ which is Turing's fixed point combinator.
- Y_2 corresponds to $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ which is Curry's fixed point combinator.
- $\lambda f.Y_3(f)$ corresponds to $\lambda f.(\lambda x.\lambda y.f(xxy))(\lambda x.\lambda y.f(xxy))N$.

□

Chapter 8

Action Calculi

We show that our framework for sharing graphs can accommodate Milner's *action calculi* [Mil96], a proposed framework for general interactive computation, by showing that our sharing theories, enriched with suitable constructs for interpreting parameterized constants called controls, are equivalent to the action calculi and their higher-order/reflexive extensions [Mil94a, Mil94b, Mif96].

The dynamics, the computational counterpart of action calculi, is then understood as rewriting systems on our theories, and interpreted as local preorders on our models. In this sense, we understand action calculi as generalized graph rewriting systems – and regard the notion of sharing as one of the fundamental concepts of action calculi.

We first review the definition of action calculi. We then extend our sharing theories to accommodate action calculi. The essential point is to introduce the parametrized operator symbols which correspond to controls. The semantic observations immediately tell us that we are dealing with the equivalent beings and there are obvious syntactic translations between sharing theories and action calculi. This observation extends to the higher-order and reflexive (cyclic) extensions of action calculi equally well.

8.1 Action Calculi: Definitions, Basics

We introduce an action calculus as a quotient of a term algebra. An action calculus is defined by a set of typed terms, an equational theory on it (theory AC) and a preorder on the equivalence classes of terms, called a *reaction relation* or *dynamics*.

We first fix a freely generated monoid $M = (M, \otimes, \epsilon)$; M's elements will be called *arities* and generators will be called *prime arities*. We also assume a set X; its elements are called *names*, and we assign a prime arity to each name. If an arity p is assigned to a name x, we write $x : p$. Further, we assume that infinitely many names are associated with each prime arity.

An action calculus $\mathsf{AC}(\mathcal{K})$ is then specified by a set $\mathcal{K}$ of *controls* (*control operators*) each of which is equipped with an *arity rule* and a reaction relation which is usually generated by a few *reaction rules*.

Definition 8.1.1 The set of *terms* over $\mathcal{K}$ is generated by the following rules.
[Raw terms]

$$a, b \ldots \quad ::= \quad \langle x \rangle \mid (x)a \mid \mathbf{id} \mid a \cdot b \mid a \otimes b \mid K(a_1, \ldots, a_k)$$

[Arity assignment (Typing)]

$$\frac{x : p}{\langle x \rangle : \epsilon \to p} \quad \frac{a : m \to n \quad x : p}{(x)a : p \otimes m \to n} \quad \frac{}{\mathbf{id}_m : m \to m} \quad \frac{a : k \to l \quad b : l \to m}{a \cdot b : k \to m}$$

$$\frac{a : k \to m \quad b : l \to n}{a \otimes b : k \otimes l \to m \otimes n} \quad \frac{a_i : m_i \to n_i \quad (i = 1, \ldots, k)}{K(a_1, \ldots, a_k) : m \to n}$$

where each control $K \in \mathcal{K}$ is equipped with arity $((m_1, n_1), \ldots, (m_k, n_k), (m, n))$ which may be subject to some side conditions; there can be some dependency between the number of arguments (k) and the arities m_i, n_i, m and n. $\Box$

We may omit the arity subscripts of terms if there is no confusion. The notions of *free* and *bound name* are defined as usual; a name x is bound in $(x)a$, and $\langle x \rangle$ is considered a free occurrence of x.The set of free names in a term a will be denoted by $fn(a)$.

Definition 8.1.2 We use the following derived notations for multiple names and a notation for derived terms (permutation).

$$\begin{array}{lcl} (x_1, \ldots, x_n)a & \triangleq & (x_1) \ldots (x_n)a \\ \langle x_1, \ldots, x_n \rangle & \triangleq & \langle x_1 \rangle \otimes \ldots \otimes \langle x_n \rangle \end{array}$$

$$\mathbf{p}_{m,n} \triangleq (\vec{x}, \vec{y})\langle \vec{y}, \vec{x} \rangle : m \otimes n \to n \otimes m$$

In $(x_1, \ldots, x_n)a$, names x_i are assumed to be all different. $\Box$

Definition 8.1.3 The equational theory AC is the set of equations upon terms generated by the following axioms.

[The strict monoidal category axioms]

A1	$a \cdot \mathbf{id} = a = \mathbf{id} \cdot a$	A4	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
A2	$a \otimes \mathbf{id}_\epsilon = a = \mathbf{id}_\epsilon \otimes a$	A5	$a \otimes (b \otimes c) = (a \otimes b) \otimes c$
A3	$\mathbf{id} \otimes \mathbf{id} = \mathbf{id}$	A6	$(a \cdot b) \otimes (a' \cdot b') = (a \otimes a') \cdot (b \otimes b')$

[The concrete axioms]

σ	$(\langle y \rangle \otimes \mathbf{id}_m) \cdot (x)a = a\{y/x\}$	$(a : m \to n)$
δ	$(x)((\langle x \rangle \otimes \mathbf{id}_m) \cdot a) = a$	$(x : p, a : p \otimes m \to n, x \notin fn(a))$
ζ	$\mathbf{p}_{k,m} \cdot (b \otimes a) = (a \otimes b) \cdot \mathbf{p}_{l,n}$	$(a : k \to l, b : m \to n)$

We call the equivalence classes of terms *actions*, for which we overload the same notations for terms and actions provided there is no ambiguity; a may represent a term or the equivalence class of the term depending on the context. □

We will write $\mathsf{AC}(\mathcal{K})(m, n)$ for the set of actions of arity $m \to n$.

Definition 8.1.4 The *action calculus* $\mathsf{AC}(\mathcal{K})$ is the equational theory given as above together with a preorder $\searrow_{m,n}$ (subscripts may be omitted) on $\mathsf{AC}(\mathcal{K})(m, n)$ for each m and n , called *reaction relation* or *dynamics*, which is closed under tensor, composition and abstraction such that identities are minimal, i.e. $\mathbf{id} \searrow a$ implies $a = \mathbf{id}$. □

Note that controls may not preserve the reaction relation. The definition of theory AC presented here is different from the original version in [Mil96] in the choice of primitives and axioms, but it is easy to verify that they give the same equational theory. Axioms A1∼A6 imply that $\mathsf{AC}(\mathcal{K})$ is a strict monoidal category whose objects are arities and arrows are actions. The concrete axioms σ and δ determine how names and abstractions work, whereas from ζ we can show that $\mathsf{AC}(\mathcal{K})$ is a symmetric monoidal category with symmetry $\mathbf{p}$.

The definition of dynamics implies that

Proposition 8.1.5 If $a \searrow b$ and $a \neq b$ then a must contain a control. □

Therefore any reaction must involve some controls.

Before proceeding to technical discussion, let us briefly recall how a simple polyadic π-calculus [Mil92a] (originally introduced by Tokoro and Honda as the ν-calculus [HT91], or the asynchronous π-calculus) can be represented as an action calculus. For details, see [Mil96].

Example 8.1.6 (the action calculus PIC)
The action calculus $\mathsf{AC}(\mathbf{box}, \mathbf{out})$, or PIC, is determined by the following data.

First, we assume that arities are freely generated from just one element - that is, we consider the monoid of the additive structure on natural numbers. Thus we have just one prime arity 1, and assume $\epsilon = 0$, $m \otimes n = m + n$. Then we introduce controls for input and output bindings:

$$\mathbf{out} : 1 + m \to 0 \qquad \frac{a : m \to n}{\mathbf{box}(a) : 1 \to n}$$

with the reaction relation generated by a reaction rule

$$(\langle x\rangle \cdot \mathbf{box}(a)) \otimes ((\langle x\rangle \otimes \mathbf{id}) \cdot \mathbf{out}) \quad \searrow \quad a.$$

$\Box$

In [Mil96] Milner has given a translation from a simple π-calculus into PIC which preserves the structural congruence and dynamics.

8.2 Action Calculi as Sharing Theories

We show that an action calculus is equivalent to a pure acyclic sharing theory enriched with parametrized operator symbols which correspond to control operators in action calculi. This is archived by observing that these two syntactic theories share the same semantic models.

Parametrized Operators

First we revise our definition of signature (Definition 2.1.1) so that we can accommodate parametrized operator symbols which correspond to control operators in action calculi.

Definition 8.2.1 (signature)
Let S be a set of sorts. An *(extended) S-sorted signature* is a set Σ of operation symbols together with an arity function assigning to each operation symbol K a list

$$((\vec{\sigma}_1, \vec{\tau}_1), \ldots, (\vec{\sigma}_r, \vec{\tau}_r), (\vec{\sigma}, \vec{\tau}))$$

of pairs of finite lists of elements of S. $\Box$

If $r = 0$, we recover the original definition of signatures of (non-parametrized) operators. Recall that the typing rule for an operator symbol $F : (\vec{\sigma}) \to (\vec{\tau})$ is given as

$$\frac{\Gamma \vdash M : (\vec{\sigma})}{\Gamma \vdash F(M) : (\vec{\tau})} \text{ operator}$$

Together with the parameters we extend this as

$$\frac{\Gamma, \vec{x}_i : \vec{\sigma}_i \vdash M_i : (\vec{\tau}_i)\ (i \leq r) \quad \Gamma \vdash M : (\vec{\sigma})}{\Gamma \vdash K((\vec{x}_1)M_1, \ldots, (\vec{x}_r)M_r \mid M) : (\vec{\tau})} \text{ operator}$$

where K is an operator with arity $((\vec{\sigma}_1, \vec{\tau}_1), \ldots, (\vec{\sigma}_r, \vec{\tau}_r), (\vec{\sigma}, \vec{\tau}))$. The axioms (Definition 2.2.3) make sense for parametrized operators (with an obvious modification on (subst)), but we need an additional axiom (α_{op}) for α-conversions of locally bound variables:

$$\boxed{\begin{array}{lll} \text{(subst)} \quad \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ K(\ldots \mid N) & = & K(\ldots \mid \mathsf{let}\ (\vec{x})\ \mathsf{be}\ M\ \mathsf{in}\ N) \\ (\alpha_{\mathrm{op}}) \quad K(\ldots, (\vec{x}_i)M_i, \ldots \mid N) & = & K(\ldots, (\vec{y}_i)M_i\{\vec{y}_i/\vec{x}_i\}, \ldots \mid N) \\ & & (\vec{y} \text{ are fresh variables}) \end{array}}$$

Both sides of axioms must have the same type under the same context.

Let us add some comments on parametrized operators. A parametrized operator K has two different sorts of arguments - one for arguments with some local quantification of names ($(\vec{x}_i)s_i$) and the other for a non-quantified argument (t). The most popular example of the former is lambda-abstraction: we can write the usual abstraction rule as

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda((x)M \mid 0) : \sigma \Rightarrow \tau}$$

instead of $\lambda x.M$. On the other hand, an application of a function symbol does not need any locally quantified arguments, like the following.

$$\frac{\Gamma \vdash M : \mathsf{nat}}{\Gamma \vdash \mathbf{succ}(\ \mid M) : \mathsf{nat}}$$

Another example is the input-binding in the (polyadic) π-calculus (recall the action calculus $\mathsf{AC}(\mathbf{box}, \mathbf{out})$ in the last section)

$$\frac{\Gamma, \vec{y} : m \vdash P : n \quad \Gamma \vdash x : p}{\Gamma \vdash \mathbf{box}((\vec{y})P \mid x) : n}$$

which is usually written as $x(\vec{y}).P$. The elimination rule for disjunction is also a good instance:

$$\frac{\Gamma, x : \sigma_1 \vdash L : \tau \quad \Gamma, y : \sigma_2 \vdash M : \tau \quad \Gamma \vdash N : \sigma_1 \vee \sigma_2}{\Gamma \vdash \mathbf{case}((x)L, (y)M \mid N) : \tau}$$

Therefore parametrized operators can be understood as term constructors which may involve the local bindings of variables. Gordon Plotkin points out that such operators are analogues of Aczel's general binding operators [Acz90].

As we noted, α-conversion on let-bound variables can be derived from other axioms. However, we need an axiom for α-conversion of locally bound names in controls ($\mathcal{K}_\alpha$), which cannot be derived from the other axioms.

Interpretation into Models

We give the semantic interpretation of parametrized operators (equivalently control constants) as natural families of functions (or natural transformations) on a cartesian-center SMC, c.f. [Gar95, Pow96a]. We extend the definition of structure (Definition 3.2.9) as follows.

For each operator symbol K with its arity rule $((m_1, n_1), \ldots, (m_k, n_k), (m, n))$, we assume a family of functions $[\![K]\!]_X : \mathcal{S}(\mathcal{F}(X) \otimes [\![m_1]\!], [\![n_1]\!]) \times \ldots \times \mathcal{S}(\mathcal{F}(X) \otimes [\![m_k]\!], [\![n_k]\!]) \longrightarrow \mathcal{S}(\mathcal{F}(X) \otimes [\![m]\!], [\![n]\!])$ natural in the parameter X in $\mathcal{C}$.

Using this extended definition, we can define the semantic interpretation of parametrized operators:

$$\begin{array}{rcl}[\![\Gamma \vdash K((\vec{x}_1)M_1, \ldots \mid N) : (\vec{\tau})]\!] & = & \mathcal{F}\Delta; (id \otimes [\![\Gamma \vdash N : (\vec{\tau})]\!]); \\ & & \quad [\![K]\!]_{[\![|\Gamma|]\!]}([\![\Gamma, \vec{x}_1 : \vec{\sigma}_1 \vdash M_1 : (\vec{\tau}_1)]\!], \ldots)\end{array}$$

The following result follows from [Pow96a]:

Theorem 8.2.2 The sharing models of a pure acyclic sharing theory gives rise to models of the action calculus with the same signature. □

Corollary 8.2.3 There are sound and complete translations between a pure acyclic sharing theory and the corresponding action calculus, which are inverse to each other. □

The detail of syntactic translations are found in [GH97].

Remark 8.2.4 The following examples show the difference between the traditional function symbols (non-parametrized operator symbols as given in Chapter 2) and our parametrized operator symbols. A function symbol F with its arity rule $((m, n))$ is equipped with a term construction

$$\frac{\Gamma \vdash M : m}{\Gamma \vdash F(\mid M) : n}$$

which is interpreted as an arrow $[\![F]\!] : [\![m]\!] \to [\![n]\!]$ in $\mathcal{S}$ of a cartesian-center SMC $\mathcal{F} : \mathcal{C} \to \mathcal{S}$. On the other hand, a parametrized operator G with its arity rule $((0, m), (0, n))$ has a corresponding term construction

$$\frac{\Gamma \vdash M : m}{\Gamma \vdash G(()M \mid 0) : n}$$

which is interpreted as a family of functions $[\![G]\!]_X : \mathcal{S}(\mathcal{F}X, [\![m]\!]) \to \mathcal{S}(\mathcal{F}X, [\![n]\!])$ natural in X in $\mathcal{C}$. If we assume the naturality in $\mathcal{S}$ then (by Yoneda) there is a bijection between these two interpretations, thus there is no significance in having a parametrized operator G. However, as the naturality is restricted to $\mathcal{C}$, $[\![G]\!]$'s interpretation is not determined by $[\![G]\!]_{[\![m]\!]}(id)$, thus has an extra generality. In the equational theory, $[\![G]\!]$ behaves like a function symbol which does not satisfy the axiom (subst). After constructing the classifying category in Chapter 3 we observed that the axiom (subst) has nothing to do with the completeness; without (subst), we get an interpretation of G rather than F, which possibly contains more junk arrows arising from the extra freedom of G over F. $\Box$

The dynamics of an action calculus has semantic interpretation in a cartesian-center SMC as a local preorder on it [Pow96a].

Theorem 8.2.5 The following are equivalent:

- dynamics on an action calculus,
- rewriting systems with the minimality condition on the corresponding sharing theory (Chapter 2) and
- local preorders with the minimality condition on the classifying category (Chapter 3).

$\Box$

8.3 Extensions

We recall the higher-order and reflexive extensions of the action calculi, and extend the correspondence between action calculi and sharing theories to these settings.

Higher-Order Action Calculi

In [Mil94a] Milner extends action calculi to *higher-order action calculi*, which are intended to be a unified framework for higher-order concurrent computation. There have been many proposed applications of the calculi to various concurrent/functional models, including higher-order extensions of process calculi, petri nets (proposed in [Mil94a]) and call-by-value/call-by-name variants of PCF [Jen95].

In this section we observe that higher-order action calculi (with a mild refinement) correspond precisely to our (pure) higher-order acyclic sharing theory because they share the same semantic models. Some of these results are reported also in [GH97].

Example 8.3.1 More complicated arities are needed for representing a higher-order extension of the action calculus, which will be described below. For such a purpose, the arities will be generated from a set of base arities ($b, \ldots$) as follows.

$$\begin{array}{lll} \text{prime arities} \quad p, q \ldots & ::= & b \mid m \Rightarrow n \\ \text{arities} \quad m, n \ldots & ::= & \epsilon \mid p \mid m \otimes n \end{array}$$

Intuitively, arities correspond to the types of the simply typed lambda calculus with (strictly associative) products. The arrow types are included in the prime arities because we need names of arrow types for representing higher-order communication.

Then we introduce two new controls. Given a set of controls $\mathcal{K}$ we define $\mathcal{K}^{\Rightarrow} \triangleq \mathcal{K} \cup \{\ulcorner - \urcorner, \mathbf{ap}\}$ where the controls $\ulcorner - \urcorner$ and $\mathbf{ap}$ are subject to the arity rules below.

$$\frac{a : m \to n}{\ulcorner a \urcorner : \epsilon \to m \Rightarrow n} \qquad \frac{}{\mathbf{ap} : (m \Rightarrow n) \otimes m \to n}$$

Now we can define the dynamics for higher-order computation on this extended action calculus AC($\mathcal{K}^{\Rightarrow}$) by the following reaction rules.

$$\begin{array}{llll} \beta & (\ulcorner a \urcorner \otimes \mathbf{id}) \cdot \mathbf{ap} & \searrow & a \\ \sigma_{\text{code}} & (\ulcorner a \urcorner \otimes \mathbf{id}) \cdot (x)b & \searrow & b\{\ulcorner a \urcorner / x\} \end{array}$$

Controls are assumed to preserve β and σ_{code} reactions. In this calculus, we can pass not only a name but also a *code* $\ulcorner a \urcorner$ of an action a as a higher-order datum (σ_{code} reaction). The code can be applied or decoded with help of $\mathbf{ap}$ (β reaction). □

Following [Mil94a], we proceed to define the higher-order action calculus. Since our concern is on the equational characterization of such a general framework of higher-order communication, we naturally regard the reaction rules β and σ as axioms rather than abstract rewriting rules – we refer [Mil94a] for some rewriting theoretical aspects of these reaction rules.

Definition 8.3.2 (higher-order action calculus)
The *higher-order action calculus* $\mathsf{HAC}(\mathcal{K})$ is obtained as the quotient of the action calculus $\mathsf{AC}(\mathcal{K}^{\Rightarrow})$ (see the last example) by axioms β, σ_{code} and η_{name} below

$$\boxed{\begin{array}{llcl} \beta & (\ulcorner a \urcorner \otimes \mathbf{id}) \cdot \mathbf{ap} & = & a \\ \sigma_{\text{code}} & (\ulcorner a \urcorner \otimes \mathbf{id}) \cdot (x)b & = & b\{\ulcorner a \urcorner / x\} \\ \eta_{\text{name}} & \ulcorner (\langle x \rangle \otimes \mathbf{id}) \cdot \mathbf{ap} \urcorner & = & \langle x \rangle \end{array}}$$

which is equipped with (the $\beta\sigma_{\text{code}}$-quotient of) the dynamics of $\mathsf{AC}(\mathcal{K}^{\Rightarrow})$. □

There is an axiom which is not included in the original version [Mil94a]: the axiom η_{name}. At first sight, the reader may think this is rather an artificial change. However, for codes we can derive the η-equation

$$\ulcorner (\ulcorner a \urcorner \otimes \mathbf{id}) \cdot \mathbf{ap} \urcorner = \ulcorner a \urcorner$$

from the β-axiom. Since codes and names are treated in the same way (as "values") in the higher-order action calculi, and since this rather weak η-axiom does not change the syntactic nature of the calculi (the conservativity from the first-order calculi still holds; see below), we decide to include it in our version. Actually this is a needed change to get clearer semantic models – with this axiom higher-order action calculi allow a good category-theoretical characterization. Note that a stronger η-axiom

$$\boxed{\eta_{\text{too-strong}} \quad \ulcorner (a \otimes \mathbf{id}) \cdot \mathbf{ap} \urcorner \ = \ a}$$

makes the calculus cartesian closed [Mil94a], thus changes the equational theory significantly. This is because $\eta_{\text{too-strong}}$ equates a value (the code of the left hand side) to a (possibly) non-value (right hand side). Since the models we are interested in may not be cartesian closed, we do not accept this axiom in general. For further discussion, see [Mil94a]. Anyway we have a better observation:

Theorem 8.3.3 The semantic models of a higher-order action calculus are given by those of the pure higher-order sharing theory with the same signature in cartesian centrally closed SMC's. □

Detail is found in [GH97]. This implies the syntactic equivalence (translations) between pure higher-order sharing theories and higher-order action calculi.

We already know that the action calculus embeds into the higher-order calculus by comparing molecular forms [Mil94a]. The semantic proof of this result is available, as we have for the pure acyclic sharing theory and pure higher-order acyclic sharing theory, by constructing a model of the higher-order theory from that of the first-order one (Chapter 5). We need a little care with parametrized operators (controls): for simplicity, we just consider the one-parameter case. Assume that $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ is a cartesian-center SMC, and $[\![-]\!]$ is a model in it, and we have an operator K of arity rule $((m_1, n_1), (m, n))$. Then there is a family of functions $[\![K]\!]_X : \mathcal{S}(\mathcal{F}(X) \otimes [\![m_1]\!], [\![n_1]\!]) \longrightarrow \mathcal{S}(\mathcal{F}(X) \otimes [\![m]\!], [\![n]\!])$ natural in X in $\mathcal{C}$. Following Corollary 5.2.2, we have a cartesian centrally closed SMC $\bar{\mathcal{F}} : \bar{\mathcal{C}} \to \bar{\mathcal{S}}$ with a fully faithful cartesian-center functor $(i_{\mathcal{C}}, i_{\mathcal{S}})$ from $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ to $\bar{\mathcal{F}} : \bar{\mathcal{C}} \to \bar{\mathcal{S}}$, where $i_{\mathcal{C}}$ is dense. Since $i_{\mathcal{S}}$ is fully faithful strict symmetric monoidal and $\bar{\mathcal{F}} \circ i_{\mathcal{C}} = i_{\mathcal{S}} \circ \mathcal{F}$, this induces a family of functions $[\![K]\!]'_X : \bar{\mathcal{S}}(\bar{\mathcal{F}}(i_{\mathcal{C}}(X)) \otimes i_{\mathcal{S}}([\![m_1]\!]), i_{\mathcal{S}}([\![n_1]\!])) \longrightarrow \bar{\mathcal{S}}(\bar{\mathcal{F}}(i_{\mathcal{C}}(X)) \otimes i_{\mathcal{S}}([\![m]\!]), i_{\mathcal{S}}([\![n]\!]))$ natural in X in $\mathcal{C}$. Since $i_{\mathcal{C}}$ is dense, we can extend $[\![K]\!]'$ to a family of functions $[\![K]\!]'_X : \bar{\mathcal{S}}(\bar{\mathcal{F}}(X) \otimes i_{\mathcal{S}}([\![m_1]\!]), i_{\mathcal{S}}([\![n_1]\!])) \longrightarrow \bar{\mathcal{S}}(\bar{\mathcal{F}}(X) \otimes i_{\mathcal{S}}([\![m]\!]), i_{\mathcal{S}}([\![n]\!]))$ natural in X in $\bar{\mathcal{C}}$. Following the same discussion in Chapter 5, we obtain the proof of conservativity.

Reflexive Action Calculi

In [Mil94b] Milner extends action calculi to allow cyclic bindings of names by introducing an operator called *reflexion*. It may be of some surprise that his notion of reflexion turns out to be equivalent to traces of symmetric monoidal categories. This observation, by Alex Mifsud and myself, first reported in Mifsud's thesis [Mif96], enables us to see that (cartesian center) traced monoidal categories serve as models of reflexive action calculi, and also to accommodate reflexive action calculi as cyclic sharing theories.

Definition 8.3.4 (reflexive action calculus)
The *reflexive action calculus* $\mathsf{AC}^r(\mathcal{K})$ [Mil94b] is obtained as the quotient of the action calculus $\mathsf{AC}(\uparrow, \mathcal{K})$ by axioms $\rho_1 \sim \rho_6$ as below.

1. The control $\uparrow$ is given by

$$\frac{a : p \otimes m \to p \otimes n}{\uparrow_p^{m,n} (a) : m \to n}$$

   (in the sequel, subscripts may be omitted). Unlike the usual controls, $\uparrow$ is assumed to preserve the reaction relation.

2. The reflexion axioms:

$$\boxed{\begin{array}{llcl}
\rho_1 & \mathbf{id}_p & = & \uparrow_p (\mathbf{p}_{p,p}) \\
\rho_2 & \uparrow_p (a) \otimes \mathbf{id} & = & \uparrow_p (a \otimes \mathbf{id}) \\
\rho_3 & \uparrow_p (a) \cdot b & = & \uparrow_p (a \cdot (\mathbf{id}_p \otimes b)) \\
\rho_4 & a \cdot \uparrow_p (b) & = & \uparrow_p ((\mathbf{id}_p \otimes a) \cdot b) \\
\rho_6 & \uparrow_q (\uparrow_p (a)) & = & \uparrow_p (\uparrow_q ((\mathbf{p}_{q,p} \otimes \mathbf{id}) \cdot a \cdot (\mathbf{p}_{p,q} \otimes \mathbf{id})))
\end{array}}$$

□

Note that we do not include the axiom ρ_5 in the original definition here, because ρ_5 was proved to be redundant by me – the proof is found in [Mif96].

In the definition above, the reflexion operator is defined only on prime arities. We extend it to general arities by

$$\begin{array}{rcl} \uparrow^{m,n}_{\epsilon} (a) & \equiv & a \\ \uparrow^{m,n}_{p\otimes l} (a) & \equiv & \uparrow^{m,n}_{l} (\uparrow^{l\otimes m, l\otimes n}_{p} (a)). \end{array}$$

For ease of explanation, we extend the definition of reflexion to a general strict symmetric monoidal category.

Definition 8.3.5 Let $\mathcal{C}$ be a strict symmetric monoidal category and assume that $Obj(\mathcal{C})$ is freely generated from a set of objects $\mathcal{P}$. A *reflexion* on $\mathcal{C}$ is a family of functions

$$\uparrow^{A,B}_{P}: \mathcal{C}(P\otimes A, P\otimes B) \longrightarrow \mathcal{C}(A,B)$$

where A, B are objects of $\mathcal{C}$ and P varies in $\mathcal{P}$, subject to the axioms $\rho_1 \sim \rho_6$. □

A simple and pleasant fact is that traces and reflexions are precisely the same:

Proposition 8.3.6 Let $\mathcal{C}$ be a strict symmetric monoidal category whose objects are freely generated from a set of objects $\mathcal{P}$. Then $\mathcal{C}$ is traced if and only if it has a reflexion. □

The proof is elementary but lengthy – see [Mif96] for the full calculation.

Corollary 8.3.7 A reflexive action calculus is traced. □

This, together with the observations on action calculi and acyclic sharing theories, immediately implies that the models of a reflexive action calculus are given by models of the corresponding pure cyclic sharing theory in cartesian-center traced SMC's. Therefore we have the correspondence between reflexive action calculi and pure cyclic sharing theories.

Combining the results above, we also have the correspondence between a higher-order reflexive action calculus and the pure higher-order cyclic sharing theory with the same signature, which share the same models in cartesian centrally closed traced SMC's.

The combination of higher-order and reflexive features enables us to encode recursive programs – this is essentially a consequence of results in Chapter 7. We give an example to motivate some intuition, as well as to make a connection with previous work by Mifsud [Mif96].

Example 8.3.8 In functional languages with a fixed-point operator Y, we can define the infinite application (loop) of a function M as $Y(\lambda x.\lambda y.M(xy))$ or $Y(\lambda x.\lambda y.x(My))$. If the language allows us to use letrec-bindings (as in cyclic sharing theories), they can be written as $\mathsf{letrec}\ x = \lambda y.M(xy)\ \mathsf{in}\ x$ and $\mathsf{letrec}\ x = \lambda y.x(My)\ \mathsf{in}\ x$ respectively. We shall demonstrate these simple forms of iteration exist in the higher-order reflexive action calculus; this is a variation of the observations in the last chapter.

Let $a : m \longrightarrow m$ be an action in the higher-order reflexive action calculus. In [Mif96] Mifsud defines actions ITER(a) and BACKITER(a) by

$$\frac{a : m \longrightarrow m \quad x : n \Rightarrow m \quad x \notin fn(a)}{\mathrm{ITER}(a) \equiv (\mathrm{rec}((x)\ulcorner(\langle x\rangle \otimes \mathbf{id}) \cdot \mathbf{ap} \cdot a\urcorner) \otimes \mathbf{id}) \cdot \mathbf{ap} : n \longrightarrow m}$$

$$\frac{a : m \longrightarrow m \quad x : m \Rightarrow n \quad x \notin fn(a)}{\mathrm{BACKITER}(a) \equiv (\mathrm{rec}((x)\ulcorner(\langle x\rangle \otimes a) \cdot \mathbf{ap}\urcorner) \otimes \mathbf{id}) \cdot \mathbf{ap} : m \longrightarrow n}$$

where $\mathrm{rec}_p(a)$ is given by

$$\frac{a : p \otimes m \longrightarrow p \otimes n}{\mathrm{rec}_p(a) \equiv \uparrow_p (a \cdot (\mathbf{copy}_p \otimes \mathbf{id}_n)) : m \longrightarrow p \otimes n}$$

They satisfy the equations below.

$$\mathrm{ITER}(a) \ = \ \mathrm{ITER}(a) \cdot a$$

$$\mathrm{BACKITER}(a) \ = \ a \cdot \mathrm{BACKITER}(a)$$

These equations are easily and intuitively verified in the higher-order cyclic sharing theories. Assume that we have a closed term $M : m \Rightarrow m$ of the higher-order cyclic sharing theory which satisfies $[\![\ \vdash M]\!] = \ulcorner a \urcorner$ where $[\![-]\!]$ indicates the translation from the higher-order cyclic sharing theory to the higher-order reflexive action calculus. Then we have

$$\mathrm{ITER}(a) \ = \ [\![\vec{z} : n \vdash \mathsf{letrec}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).M(x\vec{y})\ \mathsf{in}\ x\vec{z} : m]\!]$$

$$\mathrm{BACKITER}(a) \ = \ [\![\vec{z} : m \vdash \mathsf{letrec}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).x(M\vec{y})\ \mathsf{in}\ x\vec{z} : n]\!]$$

For instance, one can prove ITER(a) = ITER$(a) \cdot a$ as follows. In the sharing theory, we have

$$\begin{array}{lll} & \mathsf{letrec}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).M(x\vec{y})\ \mathsf{in}\ x\vec{z} & \\ = & \mathsf{letrec}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).M(x\vec{y})\ \mathsf{in}\ (\lambda(\vec{y}).M(x\vec{y}))\vec{z} & \text{(deref)} \\ = & \mathsf{letrec}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).M(x\vec{y})\ \mathsf{in}\ M(x\vec{z}) & \beta_v \\ = & M(\mathsf{letrec}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).M(x\vec{y})\ \mathsf{in}\ x\vec{z}) & (\text{app}_2) \end{array}$$

Hence

$$\begin{array}{ll} & \text{ITER}(a) \\ = & [\![(\vec{z}) : n \vdash \mathsf{letrec}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).M(x\vec{y})\ \mathsf{in}\ x\vec{z} : m]\!] \\ = & [\![(\vec{z}) : n \vdash M(\mathsf{letrec}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).M(x\vec{y})\ \mathsf{in}\ x\vec{z}) : m]\!] \\ = & ([\![\vdash M : m \Rightarrow m]\!] \otimes [\![(\vec{z}) : n \vdash \mathsf{letrec}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).M(x\vec{y})\ \mathsf{in}\ x\vec{z} : m]\!]) \cdot \mathbf{ap} \\ = & [\![(\vec{z}) : n \vdash \mathsf{letrec}\ (x)\ \mathsf{be}\ \lambda(\vec{y}).M(x\vec{y})\ \mathsf{in}\ x\vec{z} : m]\!] \cdot a \\ = & \text{ITER}(a) \cdot a. \end{array}$$

□

ITER and BACKITER are not new in the context of functional programming languages; in [Fil94], Filinski shows how to get recursion from BACKITER (the loop combinator) and first-class continuations in the call-by-value setting.

Remark 8.3.9 In his thesis [Mif96], Mifsud asks whether ITER and BACKITER satisfy some universal property. We have a partial and positive answer for his question in some specific models. First, in many domain theoretic models, in which traces are given by least fixed point operators, ITER and BACKITER are the least ones satisfying the recursive equations. On the other hand, in our **Rel**-semantics, they are represented as

$$[\![\text{ITER}(a)]\!] = \bigcup_{\substack{f : [\![n]\!] \to [\![m]\!] \\ f;[\![a]\!] = f}} f \quad \text{and} \quad [\![\text{BACKITER}(a)]\!] = \bigcup_{\substack{f : [\![m]\!] \to [\![n]\!] \\ [\![a]\!];f = f}} f \ .$$

Thus, for any $g : [\![n]\!] \longrightarrow [\![m]\!]$ (resp. $[\![m]\!] \longrightarrow [\![n]\!]$) such that $g; [\![a]\!] = g$ (resp. $[\![a]\!]; g = g$), we have $g \subseteq [\![\text{ITER}(a)]\!]$ (resp. $g \subseteq [\![\text{BACKITER}(a)]\!]$), hence they are the universal (greatest) invariants of the relation $[\![a]\!] : m \longrightarrow m$ (w.r.t. the inclusion of relations). □

Chapter 9

Conclusion

In this thesis, we have developed a theory of models of sharing graphs arising from graph rewriting theory. Generalizing the traditional theory-model correspondence between algebraic theories and finite product preserving functors, we have established the connection between theories for sharing graphs and their models described in terms of symmetric monoidal categories, strict symmetric monoidal functors and additional requirements (adjunctions, and traces). As an important case study, we have looked at recursive computation modeled in our higher-order cyclic sharing theories and their models. Also we have shown that Milner's action calculi can be understood in terms of our sharing theories enriched with parametrized operators. As an interesting implication, our axiomatic treatment of the classes of models has enabled us to compare them with those for related theories, including Moggi's notions of computation as well as intuitionistic linear type theory.

One important piece of work yet to be done is to strengthen the connection with rewriting theory. While our work establishes the equational foundation of sharing graphs, the analysis of rewriting on our theories is still to be done. However, as the interplay between the study of cartesian closed categories and the study of rewriting systems on lambda terms turned out to be fruitful, we hope that future work will show that our theories and models provide some useful insights into the rewriting theories. A recent work by Benaissa, Lascanne and Rose [BLR97] on sharing graphs is based on the idea of explicit substitutions [ACCL91]. Since the origin of explicit substitutions is closely related with categorical combinators on cartesian closed categories [Cur93], we expect that a similar story can be derived from our categorical models.

Yet another important direction to be examined is the extension to the premonoidal setting [PR96]. While symmetric monoidal structure serves well as the models of sharing graphs for which we do not assume any specific ordering of computation on resources, that seems to be too loose for modeling more sophisticated models of computation in which some ordering of computation should be specified. The most interesting situation seems to be the traced (symmetric) premonoidal categories. We have a preliminary definition of this notion and have shown the results corresponding to the structural theorem in [JSV96] and the fixed point theorem in this thesis (Theorem 7.2.1) and [Has97]. We expect that this generalized setting will be useful for analyzing recursive computation created from imperative features (like Landin's applicative order imperative fixed point operator, c.f. [FF96]), as well as the interplay between recursion and continuations [Fil94, Thi97]. We have a few preliminary observations on modeling a language with ML-like states in cartesian centrally closed traced symmetric *premonoidal* categories, where the imperative encoding of recursion can be analyzed.

Apart from these investigations into models of computation, the structures dealt with in this thesis provide us with several mathematically interesting questions, especially in connection with traced monoidal categories. For example, while the conservativity results on acyclic theories are relatively easily established (Chapter 5), we know very few result for the cyclic cases. As stated in Theorem 6.1.6, Plotkin has shown that there is a symmetric monoidal category to which we cannot add a trace without causing a collapse. He also has given counterexamples for cartesian categories (Remark 7.4.6). Yet we do not have answers to lots of similar questions, some have already been mentioned in Chapter 6 and 7. The difficulty of these problems seems to be deeply related to the difficulty of dealing with recursion semantically. We do not even know any generic way to construct traced categories (or categories with fixed point operators).

The role of action calculi in this thesis is somewhat, at least for the author, delicate. Since one may regard our sharing theories as a simplified action calculi (without parametrized controls), it is possible to view this thesis as devoted to the models of action calculi. We did not take that direction because we wanted to develop a self-contained coherent theory which is relatively independent of any specific computational interpretation. Action calculi are "calculi for interaction", but our results seem to have no direct connection with the notion of "interaction". However, we acknowledge that we got many fundamental ideas from the work on action calculi by many people [Mil96, Mif96, Gar95, HP95, Pow96a], and also hope that our results provide useful feedback to the study of action calculi.

Appendix A

Proofs

A.1 Proof of Proposition 6.1.5

(IF part) Assume $f : A \otimes U \to B \otimes U$ and $g : B \otimes U \to C \otimes U$.

$$\begin{array}{cll} & Tr^U(f); Tr^U(g) & \\ = & Tr^U(f; (Tr^U(g) \otimes id_U)) & \text{RightTightening} \\ = & Tr^U(f; Tr^U((id_B \otimes c_{U,U}); (g \otimes id_U); (id_C \otimes c_{U,U}))) & \text{Superposing} \\ = & Tr^U(f; Tr^U((id_B \otimes c_{U,U}); (g \otimes id_U))) & c_{U,U} = id_U \\ = & Tr^U(f; Tr^U(id_B \otimes c_{U,U}); g) & \text{RightTightening} \\ = & Tr^U(f; (id_B \otimes Tr^U(c_{U,U})); g) & \text{Superposing} \\ = & Tr^U(f; g) & \text{Yanking} \end{array}$$

(ONLY IF part) First, functoriality of Tr^U implies that

$$f = id_U \otimes Tr^U(f) \text{ for any } f : U \to U \ (*)$$

as

$$\begin{array}{cll} & f & \\ = & f; Tr^U(c_{U,U}) & \text{Yanking} \\ = & Tr^U((f \otimes id_U); c_{U,U}) & \text{RightTightening} \\ = & Tr^U(c_{U,U}; (id_U \otimes f)) & \\ = & Tr^U(c_{U,U}); Tr^U(c_{U,U}; (id_U \otimes f)) & \text{Yanking} \\ = & Tr^U(c_{U,U}; c_{U,U}; (id_U \otimes f)) & \text{by the functoriality} \\ = & Tr^U(id_U \otimes f)) & \\ = & id_U \otimes Tr^U(f) & \text{Superposing.} \end{array}$$

Let us consider the case of $U = V \otimes V$ and $f = c_{V,V}$. Then

$$\begin{array}{rcll} c_{V,V} & = & id_{V\otimes V} \otimes Tr^{V\otimes V}(c_{V,V}) & \text{by } (*) \\ & = & id_{V\otimes V} \otimes Tr^V(id_V) & \text{Vanishing, Yanking} \\ & = & id_{V\otimes V} & \text{by } (*). \end{array}$$

□

A.2 Proof of Theorem 7.1.1

From Trace to Fixed Point

Assume that $\mathcal{C}$ is a traced cartesian category. From the trace operator Tr, we define an operator $(-)^\dagger$ by

$$f^\dagger \; = \; Tr^X(f; \Delta_X) \; : \; A \longrightarrow X$$

for $f : A \times X \longrightarrow X$. We show that $(-)^\dagger$ satisfies conditions 1∼4. For ease of calculation, strict associativity is assumed.

Lemma A.2.1 For $f : A \times X \longrightarrow B \times X$,

$$Tr^X(f) \; = \; \langle id_A, Tr^X(f; \pi'_{B,X}; \Delta_X)\rangle; f; \pi_{B,X} : A \longrightarrow B.$$

Proof of Lemma:

$$\begin{array}{lll}
 & LHS & \\
= & Tr^X(\Delta_{A\times X}; (f;\pi_{B,X} \times f;\pi'_{B,X})) & \\
= & Tr^X(\Delta_{A\times X}; (id_{A\times X} \times f;\pi'_{B,X})); f; \pi_{B,X} & \text{R.T.} \\
= & Tr^{A\times X}((id_A \times f;\pi'_{B,X}); \Delta_{A\times X}); f; \pi_{B,X} & \text{Sliding} \\
= & Tr^{A\times X}((\Delta_A \times f;\pi'_{B,X};\Delta_X); (id_A \times c_{A,X} \times id_X)); f; \pi_{B,X} & \\
= & \Delta_A; Tr^{A\times X}(id_A \times ((id_A \times f;\pi'_{B,X};\Delta_X); (c_{A,X} \times id_X))); f; \pi_{B,X} & \text{L.T.} \\
= & \Delta_A; (id_A \times Tr^{A\times X}((id_A \times f;\pi'_{B,X};\Delta_X); (c_{A,X} \times id_X))); f; \pi_{B,X} & \text{S.P.} \\
= & \langle id_A, Tr^{A\times X}((id_A \times f;\pi'_{B,X};\Delta_X); (c_{A,X} \times id_X))\rangle; f; \pi_{B,X} & \\
= & \langle id_A, Tr^{A}(Tr^X((id_A \times f;\pi'_{B,X};\Delta_X); (c_{A,X} \times id_X)))\rangle; f; \pi_{B,X} & \text{Vanish.} \\
= & \langle id_A, Tr^{A}(Tr^X(id_A \times f;\pi'_{B,X};\Delta_X); c_{A,X})\rangle; f; \pi_{B,X} & \text{R.T.} \\
= & \langle id_A, Tr^{A}((id_A \times Tr^X(f;\pi'_{B,X};\Delta_X)); c_{A,X})\rangle; f; \pi_{B,X} & \text{S.P.} \\
= & \langle id_A, Tr^{A}(c_{A,A}; (Tr^X(f;\pi'_{B,X};\Delta_X) \times id_A))\rangle; f; \pi_{B,X} & \\
= & \langle id_A, Tr^{A}(c_{A,A}); Tr^X(f;\pi'_{B,X};\Delta_X)\rangle; f; \pi_{B,X} & \text{R.T.} \\
= & \langle id_A, Tr^X(f;\pi'_{B,X};\Delta_X)\rangle; f; \pi_{B,X} & \text{Yanking} \\
= & RHS &
\end{array}$$

□

Condition 1:

$$\begin{array}{rcll}
f^\dagger & = & Tr^X(f;\Delta_X) & \\
 & = & \langle id_A, Tr^X(f;\Delta_X;\pi'_{X,X};\Delta_X)\rangle; \langle f, f\rangle; \pi_{X,X} & \text{A.2.1} \\
 & = & \langle id_A, Tr^X(f;\Delta_X)\rangle; f & \\
 & = & \langle id_A, f^\dagger\rangle; f. &
\end{array}$$

Condition 2:

$$\begin{array}{rcll}
((g \times id_X); f)^\dagger & = & Tr^X((g \times id_X); f; \Delta) & \\
 & = & g; Tr^X(f;\Delta) & \text{L. Sliding} \\
 & = & g; f^\dagger. &
\end{array}$$

Condition 3:

$$\begin{array}{rcll}
(f;g)^\dagger & = & Tr^X(f;g;\Delta) & \\
 & = & Tr^X(f;\Delta;(g\times g)) & \\
 & = & Tr^X(f;\Delta;(id_Y\times g));g & \text{R. Tightening} \\
 & = & Tr^Y((id_A\times g);f;\Delta);g & \text{Sliding} \\
 & = & ((id_A\times g);f)^\dagger;g. &
\end{array}$$

Condition 4 (Bekič's lemma):
In the simple slice $\mathcal{C}//A$ (page 50), the condition is stated as

$$\langle f,g\rangle^\dagger \;=\; (\langle id_X,g\rangle;f)^\dagger;\langle id_X,g\rangle$$

for $f: X\times Y\to X$ and $g: X\times Y\to Y$. By Corollary 6.2.4, $\mathcal{C}//A$ is also a traced cartesian category, so it suffices to show this simpler equation from the axioms of traces.

We first calculate

$$\begin{array}{cll}
 & LHS & \\
= & Tr^{X\times Y}(\langle f,g\rangle;\Delta_{X\times Y}) & \\
= & Tr^X(Tr^Y(\langle f,g\rangle;\Delta_{X\times Y})) & \text{Vanish.} \\
= & Tr^X(\langle id_X, Tr^Y(\langle f,g\rangle;\Delta_{X\times Y};\pi'_{X\times Y\times X,Y};\Delta_Y)\rangle;\langle f,g\rangle;\Delta_{X\times Y};\pi_{X\times Y\times X,Y}) & \text{A.2.1} \\
= & Tr^X(\langle id_X, Tr^Y(g;\Delta_Y)\rangle;\langle f,g\rangle;\Delta_{X\times Y};\pi_{X\times Y\times X,Y}) & \\
= & Tr^X(\langle id_X,g^\dagger\rangle;\langle f;\Delta_X,g\rangle;(id_X\times c_{X,X})) & \\
= & Tr^X(\langle\langle id_X,g^\dagger\rangle;f;\Delta_X,\langle id_X,g^\dagger\rangle;g\rangle;(id_X\times c_{X,Y})) & \\
= & Tr^X(\langle\langle id_X,g^\dagger\rangle;f;\Delta_X,g^\dagger\rangle;(id_X\times c_{X,Y})) & \text{Cond.1} \\
= & Tr^X(\langle\langle id_X,g^\dagger\rangle;f;\Delta_X,id_X\rangle;(id_X\times c_{X,X});(id_X\times g^\dagger\times id_X)) & \\
= & Tr^X(\langle\langle id_X,g^\dagger\rangle;f;\Delta_X,id_X\rangle;(id_X\times c_{X,X}));(id_X\times g^\dagger) & \text{R.T.}
\end{array}$$

$$\begin{array}{cll}
 & RHS & \\
= & Tr^X(\langle id_X,g^\dagger\rangle;f;\Delta_X);\langle id_X,g^\dagger\rangle & \\
= & Tr^X(\langle id_X,g^\dagger\rangle;f;\Delta_X;(\Delta_X\times id_X));(id_X\times g^\dagger) & \text{R.T.} \\
= & Tr^X(\langle id_X,g^\dagger\rangle;f;\Delta_X;(id_X\times\Delta_X));(id_X\times g^\dagger) &
\end{array}$$

Therefore we have only to show

$$Tr^X(\langle\varphi,id_X\rangle;(id_X\times c_{X,X})) = Tr^X(\varphi;(id_X\times\Delta_X))$$

where $\varphi = \langle id_{A\times X}, g^\dagger\rangle; f; \Delta_X : X \longrightarrow X \times X$. This is verified by

$$\begin{array}{lll}
 & Tr^X(\langle \varphi, id_X\rangle; (id_X \times c_{X,X})) & \\
= & Tr^X(\Delta_X; (\varphi \times id_X); (id_X \times c_{X,X})) & \\
= & Tr^{X\times X}((\varphi \times id_X); (id_X \times c_{X,X}); (id_{X\times X} \times \Delta_X)) & \text{Sliding} \\
= & Tr^X(Tr^X((\varphi \times id_X); (id_X \times c_{X,X}); (id_{X\times X} \times \Delta_X))) & \text{Vanish.} \\
= & Tr^X(\varphi; Tr^X((id_X \times c_{X,X}); (id_{X\times X} \times \Delta_X))) & \text{L.T.} \\
= & Tr^X(\varphi; (id_X \times Tr^X(c_{X,X}; (id_X \times \Delta_X)))) & \text{S.P.} \\
= & Tr^X(\varphi; (id_X \times Tr^X((\Delta_X \times id_X); (id_X \times c_{X,X}); (c_{X,X} \times id_X)))) & \\
= & Tr^X(\varphi; (id_X \times \Delta_X; Tr^X(id_X \times c_{X,X}); c_{X,X})) & \text{L.\&R.T.} \\
= & Tr^X(\varphi; (id_X \times \Delta_X; (id_X \times Tr^X c_{X,X}); c_{X,X})) & \text{S.P.} \\
= & Tr^X(\varphi; (id_X \times \Delta_X; c_{X,X}) & \text{Yanking} \\
= & Tr^X(\varphi; (id_X \times \Delta_X). &
\end{array}$$

From Fixed Point to Trace

Let $\mathcal{C}$ be a cartesian category with an operator $(-)^\dagger$ which satisfies 1∼4. We define

$$Tr^X(f) \;=\; \langle id_A, (f; \pi'_{B,X})^\dagger\rangle; f; \pi_{B,X} \;:\; A \longrightarrow B$$

for $f : A \times X \longrightarrow B \times X$, and show that Tr is a trace operator. Again strict associativity is assumed.

Lemma A.2.2 $Tr^X(f) \;=\; ((id_A \times \pi'_{B,X}); f)^\dagger; \pi_{B,X}$.

Proof of Lemma:

$$\begin{array}{rcll}
Tr^X(f) & = & \langle id_A, (f; \pi'_{B,X})^\dagger\rangle; f; \pi_{B,X} & \\
 & = & \langle id_A, ((id_A \times \pi'_{B,X}); f)^\dagger\rangle; f; \pi_{B,X} & (3) \\
 & = & ((id_A \times \pi'_{B,X}); f)^\dagger; \pi_{B,X} & (1)
\end{array}$$

□

Vanishing:

$$\begin{array}{rcl}
Tr^1(f) & = & \langle id_A, (f; \pi'_{B,1})^\dagger\rangle; f; \pi_{B,1} \\
 & = & \langle id_A, !_A\rangle; f; id_B \\
 & = & id_A; f \\
 & = & f.
\end{array}$$

For $f : A \times X \times Y \longrightarrow B \times X \times Y$, we define

$$\begin{array}{rcll}
F & = & f; \pi_{B\times X,Y}; \pi'_{B,X} & : A \times X \times Y \longrightarrow X \\
G & = & f; \pi'_{B\times X,Y} & : A \times X \times Y \longrightarrow Y.
\end{array}$$

Then

$$\begin{array}{lrcll}
(*) & Tr^Y(f) & = & \langle id_{A\times X}, (f; \pi'_{B\times X,Y})^\dagger\rangle; f; \pi_{B\times X,Y} & \\
 & & = & \langle id_{A\times X}, G^\dagger\rangle; f; \pi_{B\times X,Y} & : \; A \times X \longrightarrow B \times Y.
\end{array}$$

We also note that $f; \pi'_{B,X\times Y} = \langle F, G\rangle : A \times X \times Y \longrightarrow X \times Y$. Then

$$\begin{array}{rcll} Tr^{X\times Y}(f) & = & \langle id_A, (f;\pi'_{B,X\times Y})^\dagger\rangle; f; \pi_{B,X\times Y} & \\ & = & \langle id_A, \langle F, G\rangle^\dagger\rangle; f; \pi_{B,X\times Y} & \\ & = & \langle id_A, \langle id_A, (\langle id_{A\times X}, G^\dagger\rangle; F)^\dagger\rangle; \langle \pi'_{A\times X}, G^\dagger\rangle\rangle; f; \pi_{B,X\times Y} & (4) \\ & = & \langle id_A, (\langle id_{A\times X}, G^\dagger\rangle; F)^\dagger\rangle; \langle id_{A\times X}, G^\dagger\rangle; f; \pi_{B,X\times Y} & \\ & = & \langle id_A, (Tr^Y(f); \pi'_{B,X})^\dagger\rangle; Tr^Y(f); \pi_{B,X} & (*) \\ & = & Tr^X(Tr^Y(f)). & \end{array}$$

Superposing:

$$\begin{array}{rll} & Tr^X((id_A \times c_{C,X}); (f \times g); (id_B \times c_{X,D})) & \\ = & \langle id_{A\times C}, ((id_A \times c_{C,X}); (f\times g); (id_B \times c_{X,D}); \pi'_{B\times D,X})^\dagger\rangle; & \\ & \quad (id_A \times c_{C,X}); (f \times g); (id_B \times c_{X,D}); \pi_{B\times D,X} & \\ = & \langle id_{A\times C}, ((\pi_{A,C} \times id_X); f; \pi'_{B,X})^\dagger\rangle; (id_A \times c_{C,X}); (f;\pi_{B,X} \times g) & \\ = & \langle id_{A\times C}, \pi_{A,C}; (f;\pi'_{B,X})^\dagger\rangle; (id_A \times c_{C,X}); (f; \pi_{B,X} \times g) & (2) \\ = & (\langle id_A, (f;\pi'_{B,X})^\dagger\rangle; f; \pi_{B,X}) \times g & \\ = & Tr^X(f) \times g. & \end{array}$$

Yanking:

$$\begin{array}{rcll} Tr^X(c_{X,X}) & = & \langle id_X, (c_{X,X}; \pi'_{X,X})^\dagger\rangle; c_{X,X}; \pi_{X,X} & \\ & = & \langle id_X, \pi^\dagger_{X,X}\rangle; \pi'_{X,X} & \\ & = & \pi^\dagger_{X,X} & \\ & = & \langle id_X, \pi^\dagger_{X,X}\rangle; \pi_{X,X} & (1) \\ & = & id_X. & \end{array}$$

Left Tightening:

$$\begin{array}{rcll} Tr^X((g\times id_X); f) & = & ((id_{A'} \times \pi'_{B,X}); (g \times id_X); f)^\dagger; \pi_{B,X} & \text{A.2.2} \\ & = & ((g \times id_{B\times X}); (id_A \times \pi'_{B,X}); f)^\dagger; \pi_{B,X} & \\ & = & g; ((id_A \times \pi'_{B,X}); f)^\dagger; \pi_{B,X} & (2). \end{array}$$

Right Tightening:

$$\begin{array}{rll} & Tr^X(f; (g \times id_X)) & \\ = & ((id_A \times \pi'_{B,X}); f; (g\times id_X))^\dagger; \pi_{B,X} & \text{A.2.2} \\ = & ((id_A \times g \times id_X); (id_A \times \pi'_{B,X}); f)^\dagger; (g \times id_X); \pi_{B,X} & (3) \\ = & ((id_A \times \pi'_{B',X}); f)^\dagger; \pi_{B',X}; g & \\ = & Tr^X(f); g. & \end{array}$$

Sliding:

$$\begin{array}{rcll} Tr^X(f; (id_B \times g)) & = & ((id_A \times \pi'_{B,X}); f; (id_B \times g))^\dagger; \pi_{B,X} & \\ & = & ((id_A \times id_B \times g); (id_A \times \pi'_{B,X}); f)^\dagger; (id_B \times g); \pi_{B,X} & (3) \\ & = & ((id_A \times \pi'_{B,X'}); (id_A \times g); f)^\dagger; \pi_{B,X'} & \\ & = & Tr^{X'}((id_A \times g); f). & \end{array}$$

A.3 Proof of Theorem 7.2.1

Let us write $U : \mathcal{S} \longrightarrow \mathcal{C}$ for the right adjoint of $\mathcal{F}$, and $\epsilon_X : UX \longrightarrow X$ (in $\mathcal{S}$) for the counit. By definition, we have a natural bijection $(-)^* : \mathcal{S}(A, B) \longrightarrow \mathcal{C}(A, UB)$. We also define $\theta_{A,X} : A \times UX \longrightarrow U(A \otimes X)$ in $\mathcal{C}$ by $\theta_{A,X} = (id_A \otimes \epsilon_X)^*$. Now we define $(-)^\dagger$ by

$$f^\dagger = Tr^{UX}(\mathcal{F}(\theta_{A,X}; Uf; \Delta_{UX})); \epsilon_X : A \longrightarrow X \quad \text{in } \mathcal{S}$$

for $f : A \otimes X \longrightarrow X$ in $\mathcal{S}$.

Condition 1: Following the remark after Theorem 7.2.1, we have only to show the simpler equation $(f; g)^\dagger = (g; f)^\dagger; g$ for $f : X \to Y$ and $g : Y \to X$.

$$\begin{array}{rcll}
(f; g)^\dagger & = & Tr^{UX}(\mathcal{F}(Uf; Ug; \Delta_{UX})); \epsilon_X & \\
& = & Tr^{UX}(\mathcal{F}(Uf; \Delta_{UY}; (Ug \times Ug))); \epsilon_X & \\
& = & Tr^{UX}(\mathcal{F}(Uf; \Delta_{UY}); (\mathcal{F}Ug \otimes \mathcal{F}Ug))); \epsilon_X & \\
& = & Tr^{UX}(\mathcal{F}(Uf; \Delta_{UY}); (id_{UY} \otimes \mathcal{F}Ug)); \mathcal{F}Ug; \epsilon_X & \text{R. Tightening} \\
& = & Tr^{UX}(\mathcal{F}(Uf; \Delta_{UY}); (id_{UY} \otimes \mathcal{F}Ug)); \epsilon_Y; g & \\
& = & Tr^{UY}(\mathcal{F}Ug; \mathcal{F}(Uf; \Delta_{UY})); \epsilon_Y; g & \text{Sliding} \\
& = & Tr^{UY}(\mathcal{F}(Ug; Uf; \Delta_{UY})); \epsilon_Y; g & \\
& = & (g; f)^\dagger; g. &
\end{array}$$

Condition 2:

$$\begin{array}{rcll}
((\mathcal{F}g \otimes id_X); f)^\dagger & = & Tr^{UX}(\mathcal{F}(\theta_{B,X}; U(g \otimes id); Uf; \Delta_{UX})); \epsilon_X & \\
& = & Tr^{UX}(\mathcal{F}((g \times id); \theta_{A,X}; Uf; \Delta_{UX})); \epsilon_X & \\
& = & Tr^{UX}((\mathcal{F}g \otimes id); \mathcal{F}(\theta_{A,X}; Uf; \Delta_{UX})); \epsilon_X & \\
& = & \mathcal{F}g; Tr^{UX}(\mathcal{F}(\theta_{A,X}; Uf; \Delta_{UX})); \epsilon_X & \text{L. Tightening} \\
& = & \mathcal{F}g; f^\dagger. &
\end{array}$$

In the proof, we used the naturality of θ:

$$\theta_{A,X}; U(\mathcal{F}(g) \otimes f) \;=\; (g \times Uf) : \theta_{B,Y} : A \times UX \longrightarrow U(B \otimes Y)$$

where $f : X \longrightarrow Y$ in $\mathcal{S}$ and $g : A \longrightarrow B$ in $\mathcal{C}$. This is routinely shown as

$$\begin{array}{rcl}
\theta_{A,X}; U(\mathcal{F}g \otimes f) & = & (id_A \otimes \epsilon_X)^*; (\epsilon_{A,X}; (\mathcal{F}g \otimes f))^* \\
& = & ((id_A \otimes \epsilon_X)^*; \epsilon_{A,X}; (\mathcal{F}g \otimes f))^* \\
& = & ((id_A \otimes \epsilon_X); (\mathcal{F}g \otimes f))^* \\
& = & (\mathcal{F}g \otimes \epsilon_X; f)^* \\
& = & (\mathcal{F}g \otimes \mathcal{F}((\epsilon_X; f)^*); \epsilon_X)^* \\
& = & (\mathcal{F}(g \times (\epsilon_X; f)^*); (id_B \otimes \epsilon_X))^* \\
& = & (g \times (\epsilon_X; f)^*); (id_B \times \epsilon_X)^* \\
& = & (g \times U(f)); \theta_{B,X}.
\end{array}$$

Remark A.3.1 Alternatively, after establishing the relation with higher-order cyclic sharing theories (or their fragments), we can use the equational theory as an internal language of our structure, and this theorem can be proved by equational reasoning in this language. See Example 7.3.11. □

A.4 Proof of Proposition 7.1.4

From the First Condition to the Second

Assume that the diagram

$$\begin{tikzcd}
A \times X \arrow[r, "f"] \arrow[d, "A\times h"'] & X \arrow[d, "h"] \\
A \times Y \arrow[r, "g"'] & Y
\end{tikzcd}$$

commutes. Then the following diagram

$$\begin{tikzcd}
A \times X \arrow[r, "f;\Delta;(h\times X)"] \arrow[d, "A\times h"'] & Y \times X \arrow[d, "Y\times h"] \\
A \times Y \arrow[r, "g;\Delta"'] & Y \times Y
\end{tikzcd}$$

also commutes. From the uniformity of the trace operator, we have

$$Tr^X(f;\Delta;(h \times X)) = Tr^Y(g;\Delta).$$

By Right Tightening, the left hand side is equal to $Tr^X(f;\Delta);h$. Since $f^\dagger = Tr^X(f;\Delta)$ and $g^\dagger = Tr^Y(g;\Delta)$, we get $f^\dagger;h = g^\dagger$.

From the Second to the First

Assume that the diagram

$$\begin{tikzcd}
A \times X \arrow[r, "f"] \arrow[d, "A\times h"'] & B \times X \arrow[d, "B\times h"] \\
A \times Y \arrow[r, "g"'] & B \times Y
\end{tikzcd}$$

commutes. Then the following diagram

$$\begin{tikzcd}
A \times B \times X \arrow[r, "(A\times\pi');f"] \arrow[d, "A\times B\times h"'] & B \times X \arrow[d, "B\times h"] \\
A \times B \times Y \arrow[r, "(A\times\pi');g"'] & B \times Y
\end{tikzcd}$$

also commutes. Since h satisfies the uniformity condition of the fixed point operator, so does $B \times h$ (by the lemma below). Thus we have

$$((A \times \pi');f)^\dagger;(B \times h) = ((A \times \pi');g)^\dagger.$$

Since $Tr^X(f) = ((A \times \pi');f)^\dagger;\pi$ and $Tr^Y(g) = ((A \times \pi');g)^\dagger;\pi$, we get $Tr^X(f) = Tr^Y(g)$.

Lemma A.4.1 If $h : X \longrightarrow Y$ and $h' : X' \longrightarrow Y'$ satisfy the uniformity condition on the fixed point operator, so does $h \times h' : X \times X' \longrightarrow Y \times Y'$.

<u>Proof:</u> Assume that the diagram

$$\begin{tikzcd} A \times X \times X' \arrow[r, "f"] \arrow[d, "A\times h\times h'"'] & X \times X' \arrow[d, "h\times h'"] \\ A \times Y \times Y' \arrow[r, "g"'] & Y \times Y' \end{tikzcd} \tag{A.1}$$

commutes. Our purpose is to show $f^\dagger;(h \times h') = g^\dagger$. By Bekič's lemma, this is equivalent to equations

$$((\langle A \times X, f_2^\dagger\rangle; f_1)^\dagger; h = ((\langle A \times Y, g_2^\dagger\rangle; g_1)^\dagger \tag{A.2}$$

$$((\langle A \times X', f'^\dagger_1\rangle; f_2')^\dagger; h' = ((\langle A \times Y', g'^\dagger_1\rangle; g_2')^\dagger \tag{A.3}$$

where $f_1 = f;\pi : A \times X \times X' \longrightarrow X$, $f_2 = f;\pi' : A \times X \times X' \longrightarrow X'$, $f_i' = (A \times c_{X',X}); f_i$, and so on. We shall show A.2. A.3 is proved in the same way.

By A.1, the diagrams

$$\begin{tikzcd} A \times X \times X' \arrow[r, "f_1"] \arrow[d, "A\times h\times h'"'] & X \arrow[d, "h"] \\ A \times Y \times Y' \arrow[r, "g_1"'] & Y \end{tikzcd} \tag{A.4}$$

$$\begin{tikzcd} A \times X \times X' \arrow[r, "f_2"] \arrow[d, "A\times X\times h'"'] & X' \arrow[d, "h'"] \\ A \times X \times Y' \arrow[r, "(A\times h\times Y');g_2"'] & Y' \end{tikzcd} \tag{A.5}$$

commute. From A.5 and the uniformity condition on h',

$$f_2^\dagger; h' = ((A \times h \times Y'); g_2)^\dagger.$$

By naturality, the right hand side is equal to $(A \times h); g_2^\dagger$. Thus we have a commutative diagram

$$\begin{tikzcd} A \times X \arrow[r, "f_2^\dagger"] \arrow[d, "A\times h"'] & X' \arrow[d, "h'"] \\ A \times Y \arrow[r, "g_2^\dagger"'] & Y' \end{tikzcd} \tag{A.6}$$

From A.4 and A.6,

$$\begin{tikzcd}
A \times X \arrow[r, "\langle A\times X, f_2^\dagger\rangle"] \arrow[d, "A\times h"'] & A \times X \times X' \arrow[r, "f_1"] \arrow[d, "A\times h\times h'"] & X \arrow[d, "h"] \\
A \times Y \arrow[r, "\langle A\times Y, g_2^\dagger\rangle"'] & A \times Y \times Y' \arrow[r, "g_1"'] & Y
\end{tikzcd}$$

commutes. Applying the uniformity condition on h, we obtain A.2. □

Remark A.4.2 The corresponding result for the uniformity condition on the trace operator follows trivially from Vanishing. □

A.5 Proof of Proposition 7.2.2

Assume that the diagram

$$\begin{tikzcd}
A \otimes X \arrow[r, "f"] \arrow[d, "A\otimes h"'] & X \arrow[d, "h"] \\
A \otimes Y \arrow[r, "g"'] & Y
\end{tikzcd}$$

commutes. Then the following diagram

$$\begin{tikzcd}
A \times UX \arrow[r, "\theta_{A,X}"] \arrow[d, "A\times Uh"'] & U(A\otimes X) \arrow[r, "Uf"] \arrow[d, "U(A\otimes h)"'] & UX \arrow[r, "\Delta_{UX}"] \arrow[d, "Uh"] & UX \times UX \arrow[d, "Uh\times Uh"] \\
A \times UY \arrow[r, "\theta_{A,Y}"'] & U(A\otimes Y) \arrow[r, "Ug"'] & UY \arrow[r, "\Delta_{UY}"'] & UY \times UY
\end{tikzcd}$$

commutes (in $\mathcal{C}$), which implies that

$$\begin{tikzcd}
A \otimes UX \arrow[r, "\mathcal{F}(\theta_{A,X};Uf;\Delta_{UX})"] \arrow[d, "A\otimes\mathcal{F}(Uh)"'] & UX \otimes UX \arrow[r, "\mathcal{F}(Uh)\otimes UX"] & UY \otimes UX \arrow[d, "UY\otimes\mathcal{F}(Uh)"] \\
A \otimes UY \arrow[rr, "\mathcal{F}(\theta_{A,Y};Ug;\Delta_{UY})"'] & & UY \otimes UY
\end{tikzcd}$$

also commutes. Then, by assumption, we have

$$Tr^{UX}(\mathcal{F}(\theta_{A,X};Uf;\Delta_{UX});(\mathcal{F}(Uh)\otimes UX)) = Tr^{UY}(\mathcal{F}(\theta_{A,Y};Ug;\Delta_{UY})).$$

The left hand side is equal to $Tr^{UX}(\mathcal{F}(\theta_{A,X};Uf;\Delta_{UX}));\mathcal{F}(Uh)$ (by Right Tightening). Composing with ϵ_Y from right to the both sides, we obtain $f^\dagger;h = g^\dagger$. □

Bibliography

[ACCL91] M. Abadi, L. Cardelli, P. -L. Curien and J. -J. Levy, Explicit substitutions. *Journal of Functional Programming* 1 (4), pages 375-416, 1991.

[Abr96a] S. Abramsky, Retracing some paths in process algebra. In *Proc. 7th Int. Conf. Concurrency Theory (CONCUR'96)*, Springer LNCS 1119, pages 1-17, 1996.

[Abr96b] S. Abramsky, Axioms for full abstraction and full completeness. Manuscript, LFCS, University of Edinburgh, 1996.

[Acz88] P. Aczel, *Non-Well-Founded Sets.* CSLI Lecture Notes 14, CSLI Publications, 1988.

[Acz90] P. Aczel, Replacement systems and the axiomatization of situation theory. *Situation Theory and Its Applications*, CSLI Lecture Notes 22, 1990.

[AA95] Z. M. Ariola and Arvind, Properties of a first-order functional language with sharing. *Theoretical Computer Science* 146, pages 69-108, 1995.

[AB97] Z. M. Ariola and S. Blom, Cyclic lambda calculi. To appear in *Proc. Int. Symp. Theoretical Aspects of Computer Software (TACS'97)*, Sendai, Springer LNCS, September 1997.

[AF96] Z. M. Ariola and M. Felleisen, A call-by-need lambda calculus. Technical report CIS-TR-96-97, 1996. To appear in *Journal of Functional Programming.*

[AK94] Z. M. Ariola and J. W. Klop, Cyclic lambda graph rewriting. In *Proc. 9th Symposium on Logic in Computer Science (LICS'94)*, pages 416-425, 1994.

[AK96] Z. M. Ariola and J. W. Klop, Equational term graph rewriting. 1996. *Fundamentae Infomaticae* 26(3,4), pages 207-240, 1996.

[BGHP96] A. Barber, P. Gardner, M. Hasegawa and G. Plotkin, Action calculi, the computational λ-calculus and linear logic. Draft, 1996.

[BGHP97] A. Barber, P. Gardner, M. Hasegawa and G. Plotkin, From action calculi to linear logic. Submitted, 1997.

[Bar96] A. Barber, Dual intuitionistic linear logic. Technical report ECS-LFCS-96-347, LFCS, University of Edinburgh, 1996.

[BvEG+87] H. Barendregt, M. van Eekelen, J. Glauert, J. Kennaway, M. Plasmeijer and M. Sleep, Term graph rewriting. In *Proc. Conf. Parallel Architecture and Langauges Europe (PARLE'87)*, Springer LNCS 259, pages 141-158, 1987.

[Bar84] H. Barendregt, *The Lambda Calculus: Its Syntax and Semantics.* Studies in Logic and the Foundations of Mathematics 103, North Holland, 1984.

[BM96] J. Barwise and L. Moss, *Vicious Circles.* CSLI Publications, 1996.

[BLR97] Z.-E.-A. Benaissa, P. Lascanne and K. H. Rose, Modeling sharing and recursion for weak reduction strategies using explicit substitution. Rapport de recherche No. 3092, INRIA, January 1997.

[BW96] N. Benton and P. Wadler, Linear logic, monads, and the lambda calculus. In *Proc. 11th Symposium on Logic in Computer Science (LICS'96)*, 1996.

[Ben95] N. Benton, A mixed linear non-linear logic: proofs, terms and models. In *Proc. Computer Science Logic 1994 (CSL'94)*, Springer LNCS 933, pages 121-135, 1995.

[Bie95] G. M. Bierman, What is a categorical model of intuitionistic linear logic? In *Proc. Second Int. Conf. Typed Lambda Calculi and Applications (TLCA'95)*, Edinburgh, Springer LNCS 902, pages 78-93, April 1995.

[BÉ93] S. L. Bloom and Z. Ésik, *Iteration Theories.* EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1993.

[BÉ96] S. L. Bloom and Z. Ésik, Fixed point operators on ccc's. Part I. *Theoretical Computer Science* 155, pages 1-38, 1996.

[Bra95] T. Braüner, The Girard translation extended with recursion. In *Proc. Computer Science Logic 1994 (CSL'94)*, Springer LNCS 933, pages 31-45, 1995.

[CP92] R. L. Crole and A. M. Pitts, New foundations for fixpoint computations: Fix hyperdoctrines and the fix logic. *Information and Computation* 98, pages 171-210, 1992.

[Cro93] R. Crole, *Categories for Types.* Cambridge University Press, 1993.

[Cur93] P. -L. Curien, *Categorical Combinators, Sequential Algorithms, and Functional Programming (Second Edition).* Birkhäuser, 1993.

[Day70] B. J. Day, On closed categories of functors. *Midwest Category Seminar Reports IV*, Springer LNM 137, pages 1-38, 1970.

[EK66] S. Eilenberg and G. M. Kelly, Closed categories. In *Prof. Conf. Categorical Algebra (La Jolla 1965)*, Springer, pages 421-562, 1966.

[Fil94] A. Filinski, Recursion from iteration. *Lisp and Symbolic Computation* 7(1), pages 11-38, 1994.

[Fre91] P. Freyd, Algebraically complete categories. In *Proc. 1990 Como Category Theory Conference*, Springer LNM 1144, pages 95-104, 1991.

[FF96] D. P. Friedman and M. Felleisen, *The Seasoned Schemer.* The MIT Press, 1996.

[GH97] P. Gardner and M. Hasegawa, Types and models for higher-order action calculi. To appear in *Proc. Int. Symp. Theoretical Aspects of Computer Software (TACS'97)*, Sendai, Springer LNCS, September 1997.

[Gar95] P. Gardner, A name-free account of action calculi. In *Proc. 11th Int. Conf. Mathematical Foundations of Programming Semantics (MFPS'95)*, Electronic Notes in Computer Science 1, Elsevier, March 1995.

[Gir87] J. -Y. Girard, Linear logic. *Theoretical Computer Science* 50, pages 1-102, 1987.

[Gir89] J. -Y. Girard, Geometry of interaction I: interpretation of system F. In *Logic Colloquium '88*, pages 221-260, North-Holland, 1989.

[Has97] M. Hasegawa, Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi. In *Proc. Third Int. Conf. Typed Lambda Calculi and Applications (TLCA'97)*, Nancy, Springer LNCS 1210, pages 196-213, April 1997.

[HJ95] C. Hermida and B. Jacobs, Fibrations with indeterminates: contextual and functional completeness for polymorphic lambda calculi. *Mathematical Structures in Computer Science* 5, pages 501-531, 1995.

[HP95] C. Hermida and A. J. Power, Fibrational control structures. In *Proc. 5th Int. Conf. Concurrency Theory (CONCUR'95)*, Springer LNCS 962, pages 117-129, 1995.

[HT91] K. Honda and M. Tokoro, An object calculus for asynchronous communication. In *Proc. ECOOP'91*, Springer LNCS 512, pages 133-147, 1991.

[IK86] G. B. Im and G. M. Kelly, A universal property of the convolution monoidal structure. *Journal of Pure and Applied Algebra* 43, pages 75-88, 1986.

[Jac94] B. Jacobs, Semantics of weakening and contraction. *Annals of Pure and Applied Logic* 69 (1), pages 73-106, 1994.

[Jen95] O. Jensen, Talk at Newton Institute, Cambridge. November 1995.

[JSV96] A. Joyal, R. Street and D. Verity, Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society* 119(3), pages 447-468, 1996.

[JS91] A. Joyal and R. Street, The geometry of tensor calculus I. *Advances in Mathematics* 88, pages 55-113, 1991.

[JS93] A. Joyal and R. Street, Braided tensor categories. *Advances in Mathematics* 102, pages 20-78, 1993.

[Kas95] C. Kassel, *Quantum Groups*. Graduate Texts in Mathematics 155, Springer, 1995.

[KL80] G. M. Kelly and M. L. Laplaza, Coherence for compact closed categories. *Journal of Pure and Applied Algebra* 19, pages 193-213, 1980.

[Kel72] G. M. Kelly, Many-variable functorial calculus I. In *Coherence in Categories*, Springer LNM 281, pages 66-105, 1972.

[Kel74] G. M. Kelly, Doctorinal adjunction. Springer LNM 420, pages 257-280, 1974.

[Koc70] A. Kock, Strong functors and monoidal monads. Various Publications Series 11, Aarhus Universitet, 1970.

[LS86] J. Lambek and P. J. Scott, *Introduction to Higher Order Categorical Logic*. Cambridge University Press, 1986.

[Lau93] J. Launchbury, A natural semantics for lazy evaluation. In *Proc. 21st ACM Symp. Principles of Programming Languages (POPL'93)*, pages 144-154, 1993.

[Law63] F. W. Lawvere, Functorial semantics of algebraic theories. *Proc. National Academy of Science* 50, pages 869-872, 1963.

[ML71] S. Mac Lane, *Categories for the Working Mathematician.* Graduate Texts in Mathematics 5, Springer-Verlag, 1971.

[MOTW95] J. Maraist, M. Odersky, D. Turner and P. Wadler, Call-by-name, call-by-value, call-by-need, and the linear lambda calculus. In *Proc. 11th Int. Conf. Mathematical Foundations of Programming Semantics (MFPS'95)*, Electronic Notes in Computer Science 1, Elsevier, March 1995.

[MMP95] A. Mifsud, R. Milner and A. J. Power, Control structures. In *Proc. 10th Symposium on Logic in Computer Science (LICS'95)*, 1995.

[Mif96] A. Mifsud, *Control Structures.* PhD thesis, LFCS, University of Edinburgh, 1996.

[MPW92] R. Milner, J. Parrow and D. Walker, A calculus of mobile processes, part I + II. *Information and Computation* 100 (1), pages 1-77, 1992.

[Mil89] R. Milner, *Communications and Concurrency.* Prentice Hall, 1989.

[Mil92a] R. Milner, The polyadic π-calculus: a tutorial. In F. L. Bauer et al. (eds.), *Logic and Algebra of Specification*, Springer Verlag, pages 203-246, 1992.

[Mil92b] R. Milner, Functions as processes. *Mathematical Structures in Computer Science* 2, pages 119-141, 1992.

[Mil94a] R. Milner, Higher-order action calculi. In *Proc. Computer Science Logic 1992 (CSL'92)*, Springer LNCS 832, pages 238-260, 1994.

[Mil94b] R. Milner, Action calculi V: reflexive molecular forms (with Appendix by O. Jensen). Third draft, July 1994.

[Mil96] R. Milner, Calculi for interaction. *Acta Informatica* 33(8), pages 707-737, 1996.

[Mog88] E. Moggi, Computational lambda-calculus and monads. Technical report ECS-LFCS-88-66, LFCS, University of Edinburgh, 1988.

[Mog91] E. Moggi, Notions of computation and monads. *Information and Computation* 93, pages 55-92, 1991.

[Mog97] E. Moggi, Metalanguages and applications. In A. M. Pitts and P. Dybjer (eds.), *Semantics and Logic of Computation*, Cambridge University Press, 1997.

[PR96] A. J. Power and E. P. Robinson, Premonoidal categories and notions of computation. 1996. To appear in *Mathematical Structures in Computer Science.*

[Pow96a] A. J. Power, Elementary control structures. In *Proc. 6th Int. Conf. Concurrency Theory (CONCUR'96)*, Springer LNCS 1119, pages 115-130, 1996.

[Pow96b] A. J. Power, Premonoidal categories as categories with algebraic structure. To appear, 1996.

[RT90] N. Yu. Reshetikhin and V. G. Turaev, Ribbon graphs and their invariants derived from quantum groups. *Communications in Mathematical Physics* 127, pages 1-26, 1990.

[Sel96] P. Selinger, Order-incompleteness and finite lambda models. In *Proc. 11th Symposium on Logic in Computer Science (LICS'96)*, pages 432-439, 1996.

[Shu94] M. C. Shum, Tortile tensor categories. *Journal of Pure and Applied Algebra* 93, pages 57-110, 1994.

[Sim92] A. K. Simpson, Recursive types in Kleisli categories. Manuscript, LFCS, University of Edinburgh, 1992.

[Sim93] A. K. Simpson, A characterisation of the least-fixed-point operator by dinaturality. *Theoretical Computer Science* 118, pages 301-314, 1993.

[Sim95] A. K. Simpson, Categorical completeness results for the simply-typed lambda-calculus. In *Proc. Second Int. Conf. Typed Lambda Calculi and Applications (TLCA'95)*, Edinburgh, Springer LNCS 902, pages 414-427, April 1995.

[SPvE93] M. Sleep, M. Plasmeijer and M. van Eekelen (eds.), *Term Graph Rewriting: Theory and Practice.* John Wiley & Sons, 1993.

[SD70] R. Street and E. Dubuc, Dinatural transformations. Springer Lecture Notes in Mathematics 137, pages 126-137, 1970.

[Thi97] H. Thielecke, Continuation passing style and self-adjointness. In *Proc. 2nd ACM SIGPLAN Workshop on Continuations*, Paris, January 1997.

[Tur79] D. A. Turner, A new implementation technique for applicative languages. *Software – Practice and Experience* 9, pages 31-49, 1979.

Index

λ_c-calculus, 75
  commutative, 75
λ_c-model, 74
  commutative, 74
$\lambda_{\mathrm{letrec}}$-calculus, 102
$\lambda^v_{\mathrm{letrec}}$-calculus, 102

action, 112
action calculus, 112
  higher-order, 117
  reflexive, 119
arity, 110
  prime, 110

Bekič's lemma, 90

CccSMC, 66
CcSMC, 49
CcTrSMC, 82
classifying category, 57, 68, 86
commutative monad, 72
commutative strength, 72
compact closed category, 79
computational lambda calculus, 75
  commutative, 75
control, 111
cyclic lambda calculus, 102

dependency ralation, 25
dynamics, 112

fixed point operator, 90
  diagonal property of –, 91
  dinatural property of –, 91
  uniformity of –, 92
functional completeness, 50, 66, 82

generic model, 57

homomorphism, 54
HSharingMod, 68

linear/non-linear model, 76
local preorder, 60

minimality condition, 39, 60
monoidal adjunction, 71
monoidal category, 43
  strict, 43
  symmetric, 43
    cartesian centrally closed, 64
    cartesian-center, 48
    cartesian-center traced, 82
    traced, 77
monoidal functor, 45
  strict, 46
  strong, 46
  symmetric, 46
    cartesian centrally closed, 65
    cartesian-center, 49
    cartesian-center traced, 82
    traced, 82
monoidal natural transformation, 47
  cartesian-center, 49

name, 110
normal form, 31, 37

PIC, 113

reaction relation, 112
reaction rule, 111
reflexion, 119
rewriting system, 38

sharing graph, 22
  acyclic, 25
  finite, 24
sharing model, 52
  cyclic, 84
  higher-order, 67
sharing structure, 50
  cyclic, 83

    higher-order, 66
sharing theory
    acyclic, 27
        higher-order, 63
    cyclic, 35
        higher-order, 99
SharingMod, 54
signature, 21, 113
    with •, 21
simple slice, 50
sort, 21
    higher-order, 63
strong monad, 72

tensorial strength, 72
theory AC, 111
trace, 77
traced computational model, 96

value, 64, 98