Higher-Order and Reflexive Action Calculi: Their Type Theory and Models[1]

Philippa Gardner* and Masahito Hasegawa†

*Computing Laboratory, University of Cambridge
email: Philippa.Gardner@cl.cam.ac.uk
†Research Institute for Mathematical Sciences, Kyoto University
email: hassei@kurims.kyoto-u.ac.jp

May 1998

[1]An earlier version has been published in proceedings of TACS'97 as [GH97].

Proposed Running Head:
Higher-Order and Reflexive Action Calculi

Corresponding Author:
Masahito Hasegawa
Research Institute for Mathematical Sciences
Kyoto University
Kyoto 606-8502, Japan
email: hassei@kurims.kyoto-u.ac.jp

Abstract

Action calculi have been introduced by Milner as a framework for representing models of interactive behaviour. Two natural extensions of action calculi have been proposed: the higher-order action calculi, which add higher-order features to the basic setting, and the reflexive action calculi, which allow circular bindings of processes and gives recursion in the presense of higher-order features.

In this paper, we present simple type theories for action calculi, higher-order action calculi and reflexive action calculi. We also give the categorical models of the extensions, by enriching Power's models of action calculi. As applications, we give a semantic proof of the conservativity of higher-order action calculi over action calculi, and a precise connection with Moggi's computational lambda calculus and notions of computation. We also relate the models of higher-order reflexive action calculi to models of recursive computation, by following the observation that the trace operator of Joyal, Street and Verity can be used to model recursion.

List of Symbols

$\mathbb{K}$	signature
P	set of primes
K	set of control operators
$\mathsf{AC}(\mathbb{K})$	the action calculus
$\mathsf{HAC}(\mathbb{K})$	the higher-order action calculus
$\mathsf{AC}^r(\mathbb{K})$	the reflexive action calculus
$\mathsf{T}(\mathbb{K})$	(first-order) type theory
$\mathsf{HT}(\mathbb{K})$	higher-order type theory
$\mathsf{T}^r(\mathbb{K})$	reflexive type theory
Φ	translation from action calculi to type theory
Ψ	translation from type theory to action calculi
$\mathbf{AMod}(\mathbb{K})$	the category of small action models
$\mathbf{HAMod}(\mathbb{K})$	the category of small higher-order action models
$\mathbf{AMod}^r(\mathbb{K})$	the category of small reflexive action models
$[\![\ ]\!]$	semantic interpretations in models

Contents

1 Introduction **1**

2 Preliminaries **2**
2.1 Action Calculi 2
2.2 Higher-Order Action Calculi 4
2.3 Reflexive Action Calculi 5

3 Type Theory **6**
3.1 First-Order Theory 6
3.2 Connection with Action Calculi 8
3.3 Higher-Order Theory 9
3.4 Connection with Higher-Order Action Calculi 10
3.5 Reflexive Theory 11
3.6 Connection with Reflexive Action Calculi 12

4 Categorical Models **12**
4.1 Models of The First-Order Theories 12
4.2 Higher-Order Extension 14
4.3 Reflexive Extension 16

5 Conservativity of Higher-Order Extension **18**

6 Notions of Computation **19**
6.1 The Computational Lambda Calculus 20
6.2 Comparison with λ_c-Models 21

7 Reflexion and Recursion **22**
7.1 Recursion in Higher-Order Reflexive Action Calculi 22
7.2 Recursion and Trace 23

8 Conclusions and Related Work **24**

A Graphical Presentation of Axioms for Trace **28**

B Higher-Order Reflexive Theory **29**

1 Introduction

Milner introduced *action calculi* as a framework for representing models of interactive behaviour [Mil96]. He also introduced two conservative extensions: *higher-order* action calculi [Mil94a], which add higher-order features to the basic setting, and *reflexive* action calculi [Mil94b], which allow circular construction of actions and give recursion in the presence of the higher-order features. We present simple type theories for action calculi and the extensions. We also give the categorical models of the extensions, by enriching Power's models of action calculi [Pow96]. It remains an on-going research area to fully understand the dynamics of action calculi and obtain, for example, a general theory of bisimulation.

Higher-Order Action Calculi and Notions of Computation

We solve two open problems for higher-order action calculi: the connection between higher-order action calculi and typed λ-calculi, and the identification of the categorical models for the higher-order case. Milner first postulated the relationship between higher-order action calculi and typed λ-calculi in [Mil94a]. He showed how to obtain a cartesian closed category from a higher-order action calculi with an extra axiom corresponding to η-equality. However, he also observed that this extra axiom collapses the higher-order structure, in that the conservativity result over action calculi is lost.

One important fact about higher-order action calculi is that we can only substitute names and codes of the form $\ulcorner a \urcorner$, where a is an action (or agent or process). This restriction is necessary, because arbitrary processes can cause side-effects and hence should not be duplicated nor discarded freely. We give a type-theoretic presentation of higher-order action calculi, where variables and λ-abstractions are the only terms that are substituted and the dynamics corresponds to term rewriting modulo a provable equality.

We also identify a class of categorical models of higher-order action calculi, by extending Power's models of action calculi [Pow96]. We use a cartesian category for interpreting the names and the codes, a symmetric monoidal category with a local preorder for interpreting arbitrary processes and their dynamics, and a symmetric monoidal adjunction for describing the interplay between the two categories. We give a sound interpretation of our higher-order type theory in these models. To show completeness, we require an extra axiom which corresponds to a η axiom for names.

There are two applications of these results. First, we give a simple semantic proof of the conservativity of (static) higher-order action calculi over action calculi using standard results from category theory. Second, we observe that our type theory is very similar to Moggi's *computational λ-calculus* [Mog88], and make this intuition precise by relating the class of the models of higher-order theories and Moggi's semantic framework called *notions of computation* [Mog91]. Our results also clarify the differences between the two approaches. Moggi describes computational behaviour using monads in a category, whereas the dynamics of action calculi are interpreted by preorders.

Reflexive Action Calculi, Letrec, and Trace

The type theory for the reflexive action calculi uses the calculi with recursive *letrec* binding widely used in many programming languages. In particular our presentation is strongly inspired from recent work on cyclic graph rewriting systems by Ariola et al [AA95].

Our models for the reflexive action calculi are based on the *traced monoidal categories* of Joyal, Street and Verity [JSV96], which provide a general foundation for modeling these sort of cyclic structures. Mifsud and Hasegawa have shown that the axioms for the reflexion

operator are equivalent to the axioms for the trace operator [Mif96]. Following their work we formulate our models of reflexive action calculi using traced monoidal categories.

As a most interesting use of reflexion, we study the recursive computation in the higher-order reflexive action calculi. Mifsud observes that recursive computation can be described using the reflexive higher-order action calculi [Mif96]. Hasegawa extends this work to provide a new class of semantic models for recursion in terms of traced monoidal categories [Has97a]. Here, we review Mifsud's construction using our type theory, and show how it leads to the general models for recursion.

Overview

This paper is organized as follows. In Section 2, we review action calculi, higher-order action calculi and reflexive action calculi. Section 3 describes the type theories, and Section 4 the corresponding models. We give three applications of these results. Section 5 gives a semantic proof of conservativity of higher-order action calculi over action calculi, and in Section 6 we relate our type theories with Moggi's work by comparing the semantic models. Section 7 is devoted for analyzing recursive computation in the higher-order reflexive setting which leads to Hasegawa's work on recursion and traced monoidal categories. We conclude in Section 8 by showing how our results fit the broader picture with discussions towards further research.

2 Preliminaries

In this section we review action calculi, higher-order action calculi and reflexive action calculi. Most of the ideas can be found in Milner's original work [Mil96, Mil94a, Mil94b], although there are a few novel features including an additional axiom for higher-order action calculi and the omission of a redundant axiom for reflexive action calculi.

2.1 Action Calculi

An action calculus is defined by a set of terms, an equational theory on the terms and a preorder on the equivalence classes of terms, called the *reaction relation* or the *dynamics*. It is generated from a *signature* $\mathbb{K} = (\mathsf{P}, \mathcal{K})$, which consists of a set P of *(basic) primes* denoted by $p, q, \ldots$ and a set $\mathcal{K}$ of *control operators*. Each control operator K of $\mathcal{K}$ is equipped with an arity rule $((m_1, n_1), \ldots (m_r, n_r)) \to (m, n)$, where the m's and n's are finite sequences of primes called *(tensor) arities*; we write ϵ for the empty sequence, (infix) $\otimes$ for concatenation and write M for the set of arities. We overload primes with arities of length one, and call them *prime arities*. We assume a fixed denumerable set X of *names*, each of which is equipped with a prime. We let $x, y, \ldots$ range over names, and sometimes write x^p to indicate that x has the prime arity p.

Definition 2.1 Terms $a, b, c \ldots$ are constructed from the basic operators: *identity* $\mathbf{id}_m$, *composition* $\cdot$, *tensor* $\otimes$, *permutation* $\mathbf{p}_{m,n}$, *abstraction* (x^p), *datum* $\langle x^p \rangle$ and the *control operators* K. A term a is assigned a pair of arities (m, n), which we write $a : m \to n$, using the following

rules:

$$\mathbf{id}_m : m \to m \qquad \frac{a : k \to l \quad b : l \to m}{a \cdot b : k \to m} \qquad \frac{a : k \to m \quad b : l \to n}{a \otimes b : k \otimes l \to m \otimes n}$$

$$\mathbf{p}_{m,n} : m \otimes n \to n \otimes m \qquad \frac{a : m \to n}{(x^p)a : p \otimes m \to n} \qquad \langle x^p \rangle : \epsilon \to p$$

$$\frac{a_1 : m_1 \to n_1 \quad \ldots \quad a_r : m_r \to n_r}{\mathsf{K}(a_1, \ldots, a_r) : m \to n}$$

where, in the derivation of $\mathsf{K}(a_1, \ldots, a_r)$, the arity rule of the control operator $\mathsf{K} \in \mathcal{K}$ is $((m_1, n_1), \ldots, (m_r, n_r)) \to (m, n)$. □

Notation. We omit the arity subscripts on the basic operators when apparent. The notions of *free* and *bound name* are standard: (x) binds x and $\langle x \rangle$ represents a free occurrence of x. We write $a\{b/\langle x \rangle\}$ to denote the usual capture-avoiding substitution. The set of names free in $a, b, \ldots$ is denoted by $\mathsf{fn}(a, b, \ldots)$. Given a (possibly empty) sequence of names $\vec{x} = x_1^{p_1}, \ldots, x_r^{p_r}$ (we use vectors like $\vec{x}$, $\vec{y}$ for sequences of names), we write $|\vec{x}|$ for $p_1 \otimes \ldots \otimes p_r$. All terms and expressions used are well formed, and all equations are between terms of the same arity. We let $(\vec{x})a$ denote $(x_1) \cdots (x_r)a$, where the x_i's are distinct, and let $\langle \vec{x} \rangle$ denote $\langle x_1 \rangle \otimes \cdots \otimes \langle x_r \rangle$, where $(\ )a$ is a and $\langle\ \rangle$ is $\mathbf{id}_\epsilon$. We write Δ_m for the term $(\vec{x})\langle \vec{x}\vec{x} \rangle : m \to m \otimes m$, and $\boldsymbol{\omega}_m$ for $(\vec{x})\mathbf{id}_\epsilon : m \to \epsilon$.

Definition 2.2 The equational theory AC is the congruence relation on terms generated by the axioms of a strict monoidal category with a symmetry $\mathbf{p}$:

$$\begin{array}{ll}
a \cdot \mathbf{id} = a = \mathbf{id} \cdot a & a \cdot (b \cdot c) = (a \cdot b) \cdot c \\
a \otimes \mathbf{id}_\epsilon = a = \mathbf{id}_\epsilon \otimes a & a \otimes (b \otimes c) = (a \otimes b) \otimes c \\
\mathbf{id} \otimes \mathbf{id} = \mathbf{id} & (a \cdot b) \otimes (a' \cdot b') = (a \otimes a') \cdot (b \otimes b') \\
\mathbf{p}_{m,n} \cdot (b \otimes a) = (a \otimes b) \cdot \mathbf{p}_{m',n'} & \mathbf{p}_{l \otimes m,n} = (\mathbf{id}_l \otimes \mathbf{p}_{m,n}) \cdot (\mathbf{p}_{l,n} \otimes \mathbf{id}_m) \\
\mathbf{p}_{m,n} \cdot \mathbf{p}_{n,m} = \mathbf{id}_{m \otimes n} &
\end{array}$$

together with the following two axioms (σ) and (δ):

$$\begin{array}{lrcl}
(\sigma) & (\langle y \rangle \otimes \mathbf{id}_m) \cdot (x)a & = & a\{\langle y \rangle / \langle x \rangle\} \\
(\delta) & (x)((\langle x \rangle \otimes \mathbf{id}_m) \cdot a) & = & a \quad \text{where } x \notin \mathsf{fn}(a)
\end{array}$$

□

It is an immediate consequence of these axioms that $\mathbf{id}_m = (\vec{x})\langle \vec{x} \rangle$ and $\mathbf{p}_{m,n} = (\vec{x}\vec{y})\langle \vec{y}\vec{x} \rangle$, where $|\vec{x}| = m$ and $|\vec{y}| = n$. Our axiomatization is slightly different from the original presentation [Mil96] in the choice of primitive term constructors and axioms, but it is routine to see that these two versions are equivalent.

Definition 2.3 The *actions* are given by the equivalence classes obtained by quotienting the terms (Definition 2.1) by the equational theory AC (Definition 2.2). We overload notation and use $a, b, c, \ldots$ to denote actions as well as terms. An *action calculus* $\mathsf{AC}(\mathbb{K})$ consists of the actions and a preorder $\searrow$ on actions which preserves the arities and is closed under composition, tensor and abstraction, such that $\mathbf{id} \searrow a$ implies $a = \mathbf{id}$. The preorder $\searrow$ is also called the *dynamics* or the *reaction relation* of the action calculus. □

A *static* action calculus is an action calculus with the trivial dynamics (i.e. the identity relation on actions).

Example 2.4 A simple example of non-deterministic behaviour can be described in an action calculus by a control operator $\mathsf{plus} : ((m,n),(m,n)) \to (m,n)$ with the dynamics generated by the rules:

$$\mathsf{plus}(a,b) \ \searrow \ a \quad \text{and} \quad \mathsf{plus}(a,b) \ \searrow \ b.$$

This behaviour is analogous to a choice operator $+$ in process calculi with reduction rules $P+Q \to P$ and $P+Q \to Q$. We refer to [Mil96] for more interesting examples, including the π-calculus and Petri nets, which explore the use of action calculi as a framework for interactive behaviour. The simple example given here is enough to illustrate the ideas presented in this paper. □

2.2 Higher-Order Action Calculi

Higher-order action calculi [Mil94a] extend action calculi so that actions as well as names are substituted for names. The idea is that given an action $a : m \to n$, we package up the action in a *code* $\ulcorner a \urcorner : \epsilon \to (m \Rightarrow n)$, which uses the arity $(m \Rightarrow n)$ to keep a record of the arity structure of a. Such a "closure" of an action can be copied and discarded via substitution, and unpacked using an *application* control **ap**.

Definition 2.5 The *higher-order action calculus* $\mathsf{HAC}(\mathbb{K})$, where $\mathbb{K} = (\mathsf{P}, \mathcal{K})$ is a signature, is given by extending the definition of action calculi as follows:

1. the set of *higher-order primes* and the set of *higher-order arities* are constructed from the following grammars:

(basic) primes	$p_0, q_0 \ldots$	$\in \mathsf{P}$	
higher-order primes	$p, q \ldots$	$::=$	$p_0 \mid m \Rightarrow n$
higher-order arities	$m, n \ldots$	$::=$	$p \mid m \otimes n \mid \epsilon$

2. the set of terms is generated by the rules in Definition 2.1, plus the rules

$$\frac{a : m \to n}{\ulcorner a \urcorner : \epsilon \to m \Rightarrow n} \qquad \mathbf{ap} : (m \Rightarrow n) \otimes m \to n$$

3. the equational theory is the set of equations upon terms generated from the axioms in Definition 2.2, plus the higher-order axioms

(σ_{code})	$(\ulcorner a \urcorner \otimes \mathbf{id}) \cdot (x)b$	$=$	$b\{\ulcorner a \urcorner / \langle x \rangle\}$
(β)	$(\ulcorner a \urcorner \otimes \mathbf{id}) \cdot \mathbf{ap}$	$=$	a
(η_{name})	$\ulcorner (\langle x \rangle \otimes \mathbf{id}) \cdot \mathbf{ap} \urcorner$	$=$	$\langle x \rangle$

4. the dynamic relation is defined in the same way as the dynamics for action calculi.

□

In addition to the original axioms in [Mil94a], we include the axiom η_{name}. Notice that, for a code $\ulcorner a \urcorner$, we can prove $\ulcorner (\ulcorner a \urcorner \otimes \mathbf{id}) \cdot \mathbf{ap} \urcorner = \ulcorner a \urcorner$ using the β-axiom. The axiom η_{name} states the analogous equality for higher-order names. With this axiom, names and codes thus behave in a similar fashion. In section 4, we give further justification for η_{name} by studying the models.

Remark 2.6 In [Mil94a], Milner shows that adding a "strong" η axiom of the form

$$\ulcorner (a \otimes \mathbf{id}) \cdot \mathbf{ap} \urcorner \;=\; a$$

collapses the structure, in the sense that $\mathsf{HAC}(\mathbb{K})$ with the strong η-axiom is not a conservative extension of $\mathsf{AC}(\mathbb{K})$. In section 5, we prove the conservativity result for our weaker axiom η_{name}. □

While it is possible to regard the higher-order axioms as a part of dynamics and study the properties of the reduction theory (as given in [Mil94a]), we concentrate in this paper on the study of equational theory, since our primary concern is to identify the semantic models of the calculi.

2.3 Reflexive Action Calculi

In [Mil94b] Milner extends action calculi to allow cyclic bindings of names by introducing an operator called *reflexion*.

Definition 2.7 The *reflexive action calculus* $\mathsf{AC}^r(\mathbb{K})$, where $\mathbb{K} = (\mathsf{P}, \mathcal{K})$ is a signature, is given by extending the definition of action calculi as follows:

1. the sets of primes and arities are the same as those for the basic action calculus generated from the signature $\mathbb{K}$;
2. the set of terms is generated by the rules in Definition 2.1, plus the reflexion rule

$$\frac{a : p \otimes m \to p \otimes n}{\uparrow_p^{m,n} (a) : m \to n}$$

   (in the sequel, subscripts may be omitted)

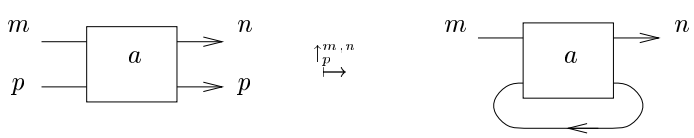
3. the equational theory is the set of equations upon terms generated from the axioms in Definition 2.2, plus the reflexion axioms:

$$\begin{array}{lrcl}
\rho_1 & \mathbf{id}_p & = & \uparrow_p (\mathbf{p}_{p,p}) \\
\rho_2 & \uparrow_p (a) \otimes \mathbf{id} & = & \uparrow_p (a \otimes \mathbf{id}) \\
\rho_3 & \uparrow_p (a) \cdot b & = & \uparrow_p (a \cdot (\mathbf{id}_p \otimes b)) \\
\rho_4 & a \cdot \uparrow_p (b) & = & \uparrow_p ((\mathbf{id}_p \otimes a) \cdot b) \\
\rho_6 & \uparrow_q (\uparrow_p (a)) & = & \uparrow_p (\uparrow_q ((\mathbf{p}_{q,p} \otimes \mathbf{id}) \cdot a \cdot (\mathbf{p}_{p,q} \otimes \mathbf{id})))
\end{array}$$

4. the dynamic relation is defined in the same way as the dynamics for action calculi, with the additional property that the relation is closed under the reflexion operator.

□

The intuition behind this definition is that reflexion creates a cyclic binding, in the sense that the construction of the term $\uparrow_p^{m,n} (a) : m \to n$ can be pictured as:



Remark 2.8 We drop an axiom included in the original axiomatization in [Mil94b]

$$\rho_5 \quad (x^q) \uparrow_p (a) = \uparrow_p ((\mathbf{p}_{p,q} \otimes \mathbf{id}) \cdot (x^q)a)$$

which has been shown to be derivable from other axioms by Hasegawa (see [Mif96]). This observation implies that the notion of reflexion is independent of the notion of names and abstractions in action calculi – none of other axioms involve names nor abstractions. In fact, the notion of reflexion turns out to make sense for any symmetric monoidal categories, as an instance of the notion of *trace* of Joyal, Street and Verity for general balanced monoidal categories. We will develop the semantics of reflexive action calculi along this line in Section 4. □

In the definition above, the reflexion operator is defined only on prime arities. We extend it to general arities by

$$\begin{array}{lcl} \uparrow_{\epsilon}^{m,n} (a) & \equiv & a \\ \uparrow_{p\otimes l}^{m,n} (a) & \equiv & \uparrow_{l}^{m,n} (\uparrow_{p}^{l\otimes m, l\otimes n} (a)). \end{array}$$

3 Type Theory

We introduce type theories corresponding to the closed fragments of action calculi and the higher-order and reflexive extensions. In particular, let(rec)-bindings are used to represent the sharing of terms and the equational theory distinguishes between those terms which can be copied and discarded at will, the *values*, and those which cannot. Since action calculi are completely determined by the closed fragment [Gar95, Gar98], our type theories are as expressive as the corresponding action calculi.

3.1 First-Order Theory

Let us fix a signature $\mathbb{K} = (\mathsf{P}, \mathcal{K})$ as introduced in the last section. The type theory $\mathsf{T}(\mathbb{K})$ consists of sequents of the form $\Gamma \vdash t : n$, where the *context* $\Gamma = x_1^{p_1}, \ldots, x_m^{p_m}$ is a sequence of distinct names (sometimes written $\vec{x}^m$), t is a term and n is an arity. This sequent corresponds to a closed action $a : |\Gamma| \to n$ in $\mathsf{AC}(\mathbb{K})$, where $|\Gamma|$ denotes $p_1 \otimes \ldots \otimes p_m$.

Definition 3.1 The set of *type-theoretic terms* (usually just called *terms*) over $\mathbb{K}$ is defined by the following grammar:

$$t \ ::= \ x \mid 0 \mid t \otimes t \mid \mathsf{let}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ t \mid \mathsf{K}((\vec{x})t, \ldots, (\vec{x})t; t)$$

where we assume that $t \otimes 0 \equiv 0 \otimes t \equiv t$ and $s \otimes (t \otimes u) \equiv (s \otimes t) \otimes u$ for terms s, t and u (strict associativity). □

The term $\mathsf{K}((\vec{x}_1)t_1, .., (\vec{x}_r)t_r; t)$ binds the variables from sequence $\vec{x}_i$ in t_i. Plotkin has pointed out that this can be viewed as a variant of Aczel's general binding operators [Acz80]. This issue is discussed in [BGHP98] and a concrete example is given in Example 3.12. The term $\mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t$ binds the variables from sequence $\vec{x}$ in t. We write $t\{u/x\}$ for the standard capture-free substitution.

Definition 3.2 A term is *well-typed* if it can be shown to annotate a sequent using the rules:

$$\Gamma, x^p \vdash x : p \qquad \Gamma \vdash 0 : \epsilon \qquad \frac{\Gamma \vdash s : m \quad \Gamma \vdash t : n}{\Gamma \vdash s \otimes t : m \otimes n} \qquad \frac{\Gamma \vdash s : m \quad \Gamma, \vec{x}^m \vdash t : n}{\Gamma \vdash \mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t : n}$$

$$\frac{\Gamma, \vec{x}_i^{m_i} \vdash s_i : n_i\ (1 \le i \le r) \quad \Gamma \vdash t : m}{\Gamma \vdash \mathsf{K}((\vec{x}_1)s_1, \ldots, (\vec{x}_r)s_r; t) : n} \qquad \frac{\Gamma, x^p, y^q, \Gamma' \vdash t : m}{\Gamma, y^q, x^p, \Gamma' \vdash t : m}$$

where, in the derivation of $\mathsf{K}(\ldots)$, K has arity rule $((m_1, n_1), \ldots, (m_r, n_r)) \to (m, n)$. □

Note that terms do not have unique type derivations. This causes a few coherence issues, see below.

The equality of the type theory should present no surprises; it gives the behaviour of the let-bindings, which describes the sharing of resources. For example, the term $\mathsf{let}\ x\ \mathsf{be}\ t\ \mathsf{in}\ x \otimes x$ describes a term t which is shared by two x's. It does not equal the term $t \otimes t$ in general.

Definition 3.3 The *equality judgement* $\Gamma \vdash s = t : n$, where $\Gamma \vdash s : n$ and $\Gamma \vdash t : n$, is defined as a congruence relation on well-typed terms of the same arity under the same context generated from the axioms:

$$\begin{array}{lcl}
\mathsf{let}\ x\ \mathsf{be}\ y\ \mathsf{in}\ t & = & t\{y/x\} \\
\mathsf{let}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ \vec{x} & = & t \\
\mathsf{let}\ \vec{x}\ \mathsf{be}\ (\mathsf{let}\ \vec{y}\ \mathsf{be}\ s\ \mathsf{in}\ t)\ \mathsf{in}\ u & = & \mathsf{let}\ \vec{y}\ \mathsf{be}\ s\ \mathsf{in}\ \mathsf{let}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ u \\
\mathsf{let}\ \vec{x},\vec{y}\ \mathsf{be}\ s \otimes t\ \mathsf{in}\ u & = & \mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ \mathsf{let}\ \vec{y}\ \mathsf{be}\ t\ \mathsf{in}\ u \\
s \otimes (\mathsf{let}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ u) & = & \mathsf{let}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ s \otimes u \\
(\mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t) \otimes u & = & \mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t \otimes u \\
\mathsf{K}((\vec{x}_1)t_1, \ldots, (\vec{x}_r)t_r; \mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t) & = & \mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ \mathsf{K}((\vec{x}_1)\mathsf{t}_1, \ldots, (\vec{x}_\mathsf{r})\mathsf{t}_\mathsf{r}; \mathsf{t})
\end{array}$$

□

Note that both sides of an axiom must have the same type under the same context. For instance, in the fourth axiom, the variables in the $\vec{x}$ cannot be free in t (written $\vec{x} \notin \mathsf{fn}(t)$), and in the last axiom $\vec{x} \notin \bigcup_{i=1}^{r} \mathsf{fn}((\vec{x}_i)t_i)$.

Remark 3.4 Our syntax allow "empty bindings" such as $\mathsf{let}\ _\ \mathsf{be}\ s\ \mathsf{in}\ t$. It follows that $\mathsf{let}\ _\ \mathsf{be}\ s\ \mathsf{in}\ t = s \otimes t = t \otimes s$, so such an empty binding has no real binding effect – the subterm s represents an isolated resource which can move around the term freely. □

Lemma 3.5 The following are derivable.

1. (simultaneous substitution)

   $$\Gamma \vdash \mathsf{let}\ \vec{x}\ \mathsf{be}\ \vec{y}\ \mathsf{in}\ t = t\{\vec{y}/\vec{x}\} : n$$

   for $\Gamma, \vec{x}^m \vdash t : n$, where $\vec{y} : m$ do not occur in Γ nor in $\vec{x}$

2. (α-conversion)

   $$\Gamma \vdash \mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t = \mathsf{let}\ \vec{y}\ \mathsf{be}\ s\ \mathsf{in}\ t\{\vec{y}/\vec{x}\} : n$$

   for $\Gamma \vdash s : m$ and $\Gamma, \vec{x}^m \vdash t : n$, where $\vec{y} : m$ do not occur in Γ nor in $\vec{x}$

3. (commutativity of independent let-bindings)

   $$\Gamma \vdash \mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ \mathsf{let}\ \vec{y}\ \mathsf{be}\ t\ \mathsf{in}\ u = \mathsf{let}\ \vec{y}\ \mathsf{be}\ t\ \mathsf{in}\ \mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ u : n$$

   for $\Gamma \vdash s : m_1$, $\Gamma \vdash t : m_2$ and $\Gamma, \vec{x}^{m_1}, \vec{y}^{m_2} \vdash u : n$

□

Remark 3.6 An alternative choice is to work with a *two-context* type theory, whose sequents have the form $\Gamma; \Sigma \vdash t : n$. Such a type theory is introduced in [BGHP96, BGHP98]. It has a direct connection with the corresponding action calculus, in that the above sequent corresponds to an action $a : |\Sigma| \to n$ with free names contained in Γ. It also gives a simpler connection with intuitionistic linear type theories with two contexts [Ben95, BP98]. From the results in [Gar95], we know that these type theories are equivalent. For our purposes, it is enough to work with the simple type theories presented here. □

3.2 Connection with Action Calculi

We give the formal justification of our assertion that the type theory $\mathsf{T}(\mathbb{K})$ corresponds to closed actions of $\mathsf{AC}(\mathbb{K})$, by giving translations between these two systems which are sound and inverse to each other.

Definition 3.7 For every $a : m \to n$ in $\mathsf{AC}(\mathbb{K})$ and sequence of distinct names $\vec{x}$ with $|\vec{x}| = m$ which are not free in a, we define a term $\Phi_{\vec{x}}(a)$ by induction on the structure of a:

$$\begin{array}{lcl}
\Phi_{\vec{x}}(\mathbf{id}_m) & = & \vec{x} \\
\Phi_{\vec{x}}(a \cdot b) & = & \mathsf{let}\ \vec{y}\ \mathsf{be}\ \Phi_{\vec{x}}(a)\ \mathsf{in}\ \Phi_{\vec{y}}(b), \quad \text{where } \vec{y} \notin \mathsf{fn}(b) \\
\Phi_{\vec{x},\vec{y}}(a \otimes b) & = & \Phi_{\vec{x}}(a) \otimes \Phi_{\vec{y}}(b) \\
\Phi_{\vec{x},\vec{y}}(\mathbf{p}_{m,n}) & = & \vec{y} \otimes \vec{x}, \quad \text{where } |\vec{x}| = m \text{ and } |\vec{y}| = n \\
\Phi_{x^p,\vec{x}}((y^p)a) & = & \Phi_{\vec{x}}(a)\{x/y\} \\
\Phi_{\emptyset}(\langle x^p \rangle) & = & x \\
\Phi_{\vec{x}}(\mathsf{K}(a_1, \ldots, a_r)) & = & \mathsf{K}((\vec{x}_1)\Phi_{\vec{x}_1}(a_1), \ldots, (\vec{x}_r)\Phi_{\vec{x}_r}(a_r); \vec{x})
\end{array}$$

□

Proposition 3.8

1. Given $a : m \to n$ such that $\mathsf{fn}(a)$ is contained in $\vec{y}$ (with $|\vec{y}| = l$), we have $\vec{y}^l, \vec{x}^m \vdash \Phi_{\vec{x}}(a) : n$.
2. Given $a = b : m \to n$ such that $\mathsf{fn}(a, b)$ is contained in $\vec{y}$ (with $|\vec{y}| = l$), we have $\vec{y}^l, \vec{x}^m \vdash \Phi_{\vec{x}}(a) = \Phi_{\vec{x}}(b) : n$. □

The translation from $\mathsf{T}(\mathbb{K})$ to $\mathsf{AC}(\mathbb{K})$ involves an inductive definition on the derivations of sequents. However, it is routine to prove a coherence result which shows that the translation is independent of the choice of derivation up to provable equality. We therefore simplify notation by working with sequents rather than derivations.

Definition 3.9 Given a sequent $\Gamma \vdash t : n$ in $\mathsf{T}(\mathbb{K})$, we define a closed term $\Psi(\Gamma \vdash t : n) : |\Gamma| \to n$ by induction on the derivation of sequents:

$$\begin{array}{lcl}
\Psi(\Gamma, x^p \vdash x : p) & = & \boldsymbol{\omega}_{|\Gamma|} \otimes \mathbf{id}_p \\
\Psi(\Gamma \vdash 0 : \epsilon) & = & \boldsymbol{\omega}_{|\Gamma|} \\
\Psi(\Gamma \vdash s \otimes t : n_1 \otimes n_2) & = & \Delta_{|\Gamma|} \cdot (\Psi(\Gamma \vdash s : n_1) \otimes \Psi(\Gamma \vdash t : n_2)) \\
\Psi(\Gamma \vdash \mathsf{let}\ \vec{y}\ \mathsf{be}\ s\ \mathsf{in}\ t : n) & = & \Delta_{|\Gamma|} \cdot (\mathbf{id}_{|\Gamma|} \otimes \Psi(\Gamma \vdash s : m)) \cdot \Psi(\Gamma, \vec{y}^m \vdash t : n) \\
\Psi(\Gamma \vdash \mathsf{K}((\vec{x}_1)s_1, \ldots; t) : n) & = & \Delta_{|\Gamma|} \cdot (\mathbf{id}_{|\Gamma|} \otimes \Psi(\Gamma \vdash t : m)) \cdot \\
 & & \quad (\vec{x})\mathsf{K}((\langle \vec{x} \rangle \otimes \mathbf{id}_{m_1}) \cdot \Psi(\Gamma, \vec{x}_1^{m_1} \vdash s_1 : n_1), \ldots) \\
\Psi(\Gamma, y^q, x^p, \Gamma' \vdash t : n) & = & (\mathbf{id}_{|\Gamma|} \otimes \mathbf{p}_{q,p} \otimes \mathbf{id}_{|\Gamma'|}) \cdot \Psi(\Gamma, x^p, y^q, \Gamma' \vdash t : n)
\end{array}$$

□

The following results state that the translations are sound and essentially inverse to each other, up to provable equality.

Proposition 3.10

1. Given $\Gamma \vdash t : n$, it follows that $\Psi(\Gamma \vdash t : n)$ is a closed term of arity $|\Gamma| \to n$.
2. Given $\Gamma \vdash s = t : n$, it follows that $\Psi(\Gamma \vdash s : n) = \Psi(\Gamma \vdash t : n)$. □

Theorem 3.11

1. Given $\Gamma \vdash t : n$, we have $\Gamma \vdash \Phi_\Gamma(\Psi(\Gamma \vdash t : n)) = t : n$ in $\mathsf{T}(\mathbb{K})$.
2. Given a closed term $a : m \to n$ and a fresh distinct sequence of names $\vec{x}^m$, we have $\Psi(\Phi_{\vec{x}}(a)) = a$. □

These results establish the connection between the *static* action calculus and our type theory. It is routine to extend them to include the dynamics. The dynamics on the type theory $\mathsf{T}(\mathbb{K})$ is a preorder $\searrow$ on the equivalence classes of the well-typed terms which respects the types and contexts, and is closed under all term constructions except controls, such that $\vec{x} \searrow t$ implies $t = \vec{x}$ and that $\vec{x} \otimes s \searrow \vec{x} \otimes t$ implies $s \searrow t$.

It is easy to see that the dynamics of an action calculus determines that of the corresponding type theory, and vice versa. Such dynamics can be regarded as a graph rewriting system on sharing graphs (see [Has97b, Mil96]).

Example 3.12 The control plus for non-determinism in Example 2.4 can be accommodated in our type theory as

$$\frac{\Gamma, \vec{x}^m \vdash s : n \quad \Gamma, \vec{y}^m \vdash t : n \quad \Gamma \vdash u : m}{\Gamma \vdash \mathsf{plus}((\vec{x})s, (\vec{y})t; u) : n}$$

with dynamics given by

$$\mathsf{plus}((\vec{x})s, (\vec{y})t; u) \searrow \mathsf{let}\ \vec{x}\ \mathsf{be}\ u\ \mathsf{in}\ s \quad \text{and} \quad \mathsf{plus}((\vec{x})s, (\vec{y})t; u) \searrow \mathsf{let}\ \vec{y}\ \mathsf{be}\ u\ \mathsf{in}\ t.$$

□

3.3 Higher-Order Theory

We extend the type theory $\mathsf{T}(\mathbb{K})$ above to incorporate higher-order features. As before, we fix a signature $\mathbb{K} = (\mathsf{P}, \mathcal{K})$.

Definition 3.13 The *higher-order theory* $\mathsf{HT}(\mathbb{K})$ is given by extending the definition of the first-order theory $\mathsf{T}(\mathbb{K})$ as follows:

1. the types are the higher-order arities given in Definition 2.5;
2. the terms are those for $\mathsf{T}(\mathbb{K})$ plus *lambda abstraction* $\lambda\vec{x}.t$ and *application* st where s and t are terms; the associated typing rules are

$$\frac{\Gamma, \vec{x}^m \vdash t : n}{\Gamma \vdash \lambda\vec{x}.t : m \Rightarrow n} \qquad \frac{\Gamma \vdash s : m \Rightarrow n \quad \Gamma \vdash t : m}{\Gamma \vdash st : n}$$

3. the equality judgement is defined as Definition 3.3, with additional axioms:

$$\begin{array}{lllll}
(\sigma_v) & \mathsf{let}\ x\ \mathsf{be}\ \lambda\vec{y}.s\ \mathsf{in}\ t & = & t\{\lambda\vec{y}.s/x\} \\
(\beta) & (\lambda\vec{x}.s)t & = & \mathsf{let}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ s \\
(\eta_0) & \lambda\vec{x}.y\vec{x} & = & y \\
 & & & \\
 & (\mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t)u & = & \mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ tu \\
 & s(\mathsf{let}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ u) & = & \mathsf{let}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ su
\end{array}$$

□

Remark 3.14 Note that we allow "empty abstraction" like $\lambda.t$, and our axioms imply that $(\lambda.t)s = \mathsf{let}\ _\ \mathsf{be}\ s\ \mathsf{in}\ t = s \otimes t = t \otimes s$ for $s : \epsilon$ (in particular $(\lambda.t)0 = t$) as well as $\lambda.y0 = y$. Such an empty abstraction is used in Section 7 for creating recursion under the presence of reflexion. □

Let us state a few immediate consequences of these axioms:

Lemma 3.15 The following equalities are derivable.

1. (α-conversion)
$$\Gamma \vdash \lambda\vec{x}.t = \lambda\vec{y}.t\{\vec{y}/\vec{x}\} : m \Rightarrow n$$
for $\Gamma, \vec{x}^m \vdash t : n$, where $\vec{y} : m$ do not occur in Γ nor in $\vec{x}$

2. (call-by-value β)
$$\Gamma \vdash (\lambda\vec{x}.s)\vec{v} = s\{\vec{v}/\vec{x}\} : n$$
for $\Gamma, \vec{x}^m \vdash s : n$ and $\Gamma \vdash \vec{v} : m$, where $\vec{v}$ is a tensor product of variables and lambda abstractions

3. (call-by-value η)
$$\Gamma \vdash \lambda\vec{y}.v\vec{y} = v : m \Rightarrow n$$
for $\Gamma \vdash v : m \Rightarrow n$, where v is either a variable or a lambda abstraction

□

3.4 Connection with Higher-Order Action Calculi

It is straightforward to extend the translations between $\mathsf{T}(\mathbb{K})$ and the closed actions of $\mathsf{AC}(\mathbb{K})$ to translations between $\mathsf{HT}(\mathbb{K})$ and the closed actions of $\mathsf{HAC}(\mathbb{K})$. These translations are sound and essentially inverse to each other. We define the translations below.

Definition 3.16
For every action $a : m \to n$ in $\mathsf{HAC}(\mathbb{K})$ and a sequence of distinct names $\vec{x}$ with $|\vec{x}| = m$ which are not free in a, we define a term $\Phi_{\vec{x}}(a)$ by induction on the structure of a, using definition 3.7 and the additional cases:

$$\begin{array}{lcl} \Phi_{\emptyset}(\ulcorner a \urcorner) & = & \lambda\vec{x}.\Phi_{\vec{x}}(a) \ \text{ where } \vec{x} \notin \mathsf{fn}(a) \\ \Phi_{y,\vec{x}}(\mathbf{ap}) & = & y\vec{x} \end{array}$$

Given a sequent $\Gamma \vdash t : n$ in $\mathsf{HT}(\mathbb{K})$, we define a closed action $\Psi(\Gamma \vdash t : n) : |\Gamma| \to n$ by induction on the derivation of sequents, using definition 3.9 and the additional cases:

$$\begin{array}{lcl} \Psi(\Gamma \vdash \lambda\vec{y}.t : m \Rightarrow n) & = & (\vec{x})^{\ulcorner}(\langle\vec{x}\rangle \otimes \mathbf{id}_m) \cdot \Psi(\Gamma, \vec{y}^m \vdash t : n)^{\urcorner} \ \text{ where } |\vec{x}| = |\Gamma| \\ \Psi(\Gamma \vdash st : n) & = & \Delta_{|\Gamma|} \cdot (\Psi(\Gamma \vdash s : m \Rightarrow n) \otimes \Psi(\Gamma \vdash t : m)) \cdot \mathbf{ap} \end{array}$$

□

It is easy to extend the results given in Proposition 3.8, Proposition 3.10 and Theorem 3.11 to the higher-order setting.

3.5 Reflexive Theory

In the similar manner we extend the type theory $\mathsf{T}(\mathbb{K})$ to accommodate reflexion. A signature $\mathbb{K} = (\mathsf{P}, \mathcal{K})$ is fixed as before.

Definition 3.17 The *reflexive theory* $\mathsf{T}^r(\mathbb{K})$ is given by modifying the definition of the type theory as follows:

1. the types are the arities;
2. the terms are generated from the following grammar:

$$t \ ::= \ x \mid 0 \mid t \otimes t \mid \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ t \mid \mathsf{K}((\vec{x})t, \ldots, (\vec{x})t; t)$$

3. the well-typed terms are defined as those of $\mathsf{T}(\mathbb{K})$, except that we replace the let-binding by the *letrec* binding:

$$\frac{\Gamma, \vec{x}^m \vdash s : m \quad \Gamma, \vec{x}^m \vdash t : n}{\Gamma \vdash \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t : n}$$

4. the *equality judgement* is defined as before from the following axioms.

$$\begin{array}{lcl}
\mathsf{letrec}\ x, \vec{y}\ \mathsf{be}\ z \otimes s\ \mathsf{in}\ t & = & \mathsf{letrec}\ \vec{y}\ \mathsf{be}\ s\{z/x\}\ \mathsf{in}\ t\{z/x\} \quad (x \not\equiv z) \\
\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ \vec{x} & = & t \\
\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ (\mathsf{letrec}\ \vec{y}\ \mathsf{be}\ s\ \mathsf{in}\ t)\ \mathsf{in}\ u & = & \mathsf{letrec}\ \vec{x}, \vec{y}\ \mathsf{be}\ t \otimes s\ \mathsf{in}\ u \\
\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ \mathsf{letrec}\ \vec{y}\ \mathsf{be}\ t\ \mathsf{in}\ u & = & \mathsf{letrec}\ \vec{x}, \vec{y}\ \mathsf{be}\ s \otimes t\ \mathsf{in}\ u \\
s \otimes (\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ u) & = & \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ s \otimes u \\
(\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t) \otimes u & = & \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t \otimes u \\
\mathsf{letrec}\ \vec{x}, \vec{y}, \vec{z}\ \mathsf{be}\ s_1 \otimes s_2 \otimes s_3\ \mathsf{in}\ t & = & \mathsf{letrec}\ \vec{y}, \vec{x}, \vec{z}\ \mathsf{be}\ s_2 \otimes s_1 \otimes s_3\ \mathsf{in}\ t \\
\mathsf{K}((\vec{x}_1)t_1, \ldots, (\vec{x}_r)t_r; \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t) & = & \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ \mathsf{K}((\vec{x}_1)t_1, \ldots, (\vec{x}_r)t_r; t)
\end{array}$$

□

Again we note that both sides of an axiom must have the same type under the same context. As for the let-binding in the non-reflexive theory, α-conversion of letrec-bound variables is derivable:

Lemma 3.18 (α-conversion) If $\Gamma, \vec{x}^m \vdash s : m$ and $\Gamma, \vec{x}^m \vdash t : n$ and $\vec{y} \notin \Gamma \cup \vec{x}$, then:

$$\Gamma \vdash \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t \ = \ \mathsf{letrec}\ \vec{y}\ \mathsf{be}\ s\{\vec{y}/\vec{x}\}\ \mathsf{in}\ t\{\vec{y}/\vec{x}\} : n.$$

□

Example 3.19 Following Milner [Mil94b], we can regard a term with a trivial circularity ($\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ \vec{x}\ \mathsf{in}\ t$) as a term t with locally declared variables $\vec{x}$, and write ($\mathsf{local}\ \vec{x}\ \mathsf{in}\ t$) for this term. As a simple example, we consider the reflexive action calculus with plus (Example 3.12). We have a term

$$\frac{\Gamma, \vec{x}^m \vdash s : m \quad \Gamma, \vec{y}^m \vdash t : m}{\Gamma \vdash \mathsf{letrec}\ \vec{z}\ \mathsf{be}\ \mathsf{plus}((\vec{x})s, (\vec{y})t; \vec{z})\ \mathsf{in}\ \vec{z} : m}$$

which can react as

$$\mathsf{letrec}\ \vec{z}\ \mathsf{be}\ \mathsf{plus}((\vec{x})s, (\vec{y})t; \vec{z})\ \mathsf{in}\ \vec{z} \quad \searrow \quad \mathsf{local}\ \vec{x}\ \mathsf{in}\ s \ \ \text{or} \ \ \mathsf{local}\ \vec{y}\ \mathsf{in}\ t$$

□

3.6 Connection with Reflexive Action Calculi

The essential point in giving the translations between a reflexive action calculus and our type theory is the observation that letrec-binding and reflexion are "mutually definable":

Definition 3.20 For every $a : m \to n$ in $\mathsf{AC}^r(\mathbb{K})$ and sequence of distinct names $\vec{x}$ with $|\vec{x}| = m$ which are not free in a, we define a term $\Phi_{\vec{x}}(a)$ by induction on the structure of a as in definition 3.7 except the following cases:

$$\begin{array}{lcl} \Phi_{\vec{x}}(a \cdot b) & = & \mathsf{letrec}\ \vec{y}\ \mathsf{be}\ \Phi_{\vec{x}}(a)\ \mathsf{in}\ \Phi_{\vec{y}}(b), \quad \text{where } \vec{y} \notin \mathsf{fn}(b) \\ \Phi_{\vec{x}}(\uparrow_p (a)) & = & \mathsf{letrec}\ y, \vec{x}\ \mathsf{be}\ \Phi_{y,\vec{x}}(a)\ \mathsf{in}\ \vec{x}, \quad \text{where } y \notin \mathsf{fn}(b) \end{array}$$

Given a sequent $\Gamma \vdash t : n$ in $\mathsf{T}^r(\mathbb{K})$, we define a closed action $\Psi(\Gamma \vdash t : n) : |\Gamma| \to n$ by induction on the derivation of sequents, using definition 3.9 except the case of the letrec binding:

$$\begin{array}{l} \Psi(\Gamma \vdash \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t : n) = \\ \qquad \Delta_{|\Gamma|} \cdot (\mathbf{id}_{|\Gamma|} \otimes \uparrow_m (\mathbf{p}_{m,|\Gamma|} \cdot \Psi(\Gamma, \vec{x}^m \vdash s : m) \cdot \Delta_m)) \cdot \Psi(\Gamma, \vec{x}^m \vdash t : n) \end{array}$$

□

Again it is easy to extend the results given in Proposition 3.8, Proposition 3.10 and Theorem 3.11 to the reflexive setting.

4 Categorical Models

We base our models of higher-order and reflexive action calculi on those of action calculi proposed by Power [Pow96]. Power's models consist of a cartesian category for interpreting the names, a symmetric monoidal category for interpreting arbitrary actions, and a functor for relating the cartesian category with the symmetric monoidal category. We extend Power's idea to the higher-order case, where the cartesian category interprets the values (the names and codes), and adjunctions describe the interplay between the two categories. We also extend to the reflexive case, by adding the notion of a trace operator due to Joyal, Street and Verity [JSV96] to the symmetric monoidal category. Our approach enables us to describe the models of action calculi and the higher-order and reflexive extensions in a simple uniform way. We also note that similar structure has been used for describing models of intuitionistic linear type theory [Ben95, BP98] in a slightly different but related context (see [BGHP96], [BGHP98] and Section 8).

4.1 Models of The First-Order Theories

First we describe the models for action calculi, which we call *action models*. The *carrier* of an action model is a triple $(\mathcal{C}, \mathcal{S}, \mathcal{F})$, where $\mathcal{C}$ is a strict cartesian category, $\mathcal{S}$ is a strict symmetric monoidal category, and $\mathcal{F} : \mathcal{C} \to \mathcal{S}$ is an identity-on-objects strict symmetric monoidal functor. Therefore we identify $\mathrm{Obj}(\mathcal{C})$ with $\mathrm{Obj}(\mathcal{S})$: the functor $\mathcal{F}$ maps the terminal object of $\mathcal{C}$ to the unit object of $\mathcal{S}$, and binary products to monoidal products, preserving the symmetric monoidal structure. In addition, each action model is specified by a signature $\mathbb{K}$ and provides an interpretation function of the prime arities as objects of $\mathcal{C}$, and of control operators as natural transformations.

Remark 4.1 In [Pav97], Pavlovic gives related models of action calculi, consisting of a single symmetric monoidal category with a sub-cartesian category. □

Definition 4.2 An *action model* over signature $\mathbb{K}$, denoted by $\mathcal{A}$, is a carrier $(\mathcal{C}, \mathcal{S}, \mathcal{F})$ equipped with a function $[\![_]\!]_{\mathsf{P}} : \mathsf{P} \to \mathrm{Obj}(\mathcal{C})$, and for each operator K with the arity rule $((m_1, n_1), \ldots, (m_r, n_r)) \to (m, n)$, a natural transformation

$$[\![\mathsf{K}]\!]_{\mathcal{K}} : \mathcal{S}(\mathcal{F}(_) \otimes [\![m_1]\!], [\![n_1]\!]) \times \ldots \times \mathcal{S}(\mathcal{F}(_) \otimes [\![m_r]\!], [\![n_r]\!]) \to \mathcal{S}(\mathcal{F}(_) \otimes [\![m]\!], [\![n]\!])$$

where $[\![m]\!]$ is defined inductively by $[\![\epsilon]\!] = I$ (the terminal object of $\mathcal{C}$, equivalently the unit object of $\mathcal{S}$) and $[\![p \otimes m]\!] = [\![p]\!]_{\mathsf{P}} \otimes [\![m]\!]$. An action model is *faithful* if the functor $\mathcal{F}$ is faithful. It is *small* if the categories $\mathcal{C}$, $\mathcal{S}$ are small. □

Notation. Where convenient, we omit the subscripts from $[\![_]\!]_{\mathsf{P}}$ and $[\![_]\!]_{\mathcal{K}}$. We sometimes write $[\![_]\!]^{\mathcal{A}}$ to emphasise the particular action model $\mathcal{A}$ under consideration.

Definition 4.3 An *action morphism* $f : \mathcal{A}_1 \to \mathcal{A}_2$ between two action models over signature $\mathbb{K}$ is a pair (f_c, f_s) where $f_c : \mathcal{C}_1 \to \mathcal{C}_2$ is a functor preserving finite products strictly, and $f_s : \mathcal{S}_1 \to \mathcal{S}_2$ is a strict symmetric monoidal functor such that $\mathcal{F}_2 \circ f_c = f_s \circ \mathcal{F}_1$, for each $p \in \mathsf{P}$ we have $f_c([\![p]\!]_{\mathsf{P}}^{\mathcal{A}_1}) = [\![p]\!]_{\mathsf{P}}^{\mathcal{A}_2}$, and, for each operator K with arity rule $((m_1, n_1), \ldots, (m_r, n_r)) \to (m, n)$, we have $f_s(([\![\mathsf{K}]\!]_{\mathcal{K}}^{\mathcal{A}_1})_{(_)}(..)) = ([\![\mathsf{K}]\!]_{\mathcal{K}}^{\mathcal{A}_2})_{f_c(_)}(f_s(..))$. □

The *category of small action models*, $\mathbf{AMod}(\mathbb{K})$, is the category whose objects are the small action models, and whose morphisms are the action morphisms, with the obvious identities and composition.

Now we give the interpretation of the type theory $\mathsf{T}(\mathbb{K})$ in an arbitrary action model $\mathcal{A}$, and state the soundness and completeness results. Given the interpretation of arities $[\![_]\!] : \mathsf{M} \to \mathsf{Obj}(\mathcal{C})$, we extend this to contexts by defining $[\![\Gamma]\!] = [\![|\Gamma|]\!]$.

Definition 4.4 Given type theory $\mathsf{T}(\mathbb{K})$ and action model $\mathcal{A}$, the *interpretation* $[\![\ _\ ]\!]$ of sequents $\Gamma \vdash t : m$ in the type theory as morphisms $[\![\Gamma]\!] \to [\![m]\!]$ in $\mathcal{S}$ is defined by induction on the derivation of sequents:

$$\begin{array}{lcl}
[\![\Gamma, x^p \vdash x : p]\!] & = & \mathcal{F}(\pi'_{[\![\Gamma]\!],[\![p]\!]}) \\
[\![\Gamma \vdash 0 : \epsilon]\!] & = & \mathcal{F}(!_{[\![\Gamma]\!]}) \\
[\![\Gamma \vdash s \otimes t : m \otimes n]\!] & = & \mathcal{F}(\Delta_{[\![\Gamma]\!]}); ([\![\Gamma \vdash s : m]\!] \otimes [\![\Gamma \vdash t : n]\!]) \\
[\![\Gamma \vdash \mathsf{let}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t : n]\!] & = & \mathcal{F}(\Delta_{[\![\Gamma]\!]}); (id_{[\![\Gamma]\!]} \otimes [\![\Gamma \vdash s : m]\!]); [\![\Gamma, \vec{x}^m \vdash t : n]\!] \\
[\![\Gamma \vdash \mathsf{K}((\vec{x}_1)s_1, \ldots, (\vec{x}_r)s_r; t) : n]\!] & = & \mathcal{F}(\Delta_{[\![\Gamma]\!]}); (id_{[\![\Gamma]\!]} \otimes [\![\Gamma \vdash t : m]\!]); \\
 & & \quad [\![\mathsf{K}]\!]_{[\![\Gamma]\!]}([\![\Gamma, \vec{x}_1^{m_1} \vdash s_1 : n_1]\!], \ldots, [\![\Gamma, \vec{x}_r^{m_r} \vdash s_r : n_r]\!]) \\
[\![\Gamma, y^q, x^p, \Gamma' \vdash t : n]\!] & = & (id_{[\![\Gamma]\!]} \otimes c_{[\![q]\!],[\![p]\!]} \otimes id_{[\![\Gamma']\!]}); [\![\Gamma, x^p, y^q, \Gamma' \vdash t : n]\!]
\end{array}$$

where π', $!$ and Δ are the projection, terminal map and diagonal map in the cartesian category $\mathcal{C}$ respectively, and c is the symmetry of $\mathcal{S}$. □

The proof of the soundness of the interpretation is routine. Completeness is proved by defining a term model constructed from $\mathsf{T}(\mathbb{K})$ (or the closed fragment of $\mathsf{AC}(\mathbb{K})$ [Gar95, Pow96]). The basic idea is that the morphisms in the cartesian category $\mathcal{C}$ are constructed from (the equivalence classes of) sequents of the form $\Gamma \vdash \vec{x} : m$ and the symmetric monoidal category $\mathcal{S}$ is constructed from (the equivalence classes of) arbitrary sequents $\Gamma \vdash t : m$. The functor $\mathcal{F}$ is the obvious inclusion functor from $\mathcal{C}$ to $\mathcal{S}$. The interpretation $[\![_]\!]_{\mathsf{P}}$ and $[\![_]\!]_{\mathcal{K}}$ are then determined routinely.

Theorem 4.5

1. (Soundness) Given $\Gamma \vdash s = t : n$ in $\mathsf{T}(\mathbb{K})$, we have $[\![\Gamma \vdash s : n]\!] = [\![\Gamma \vdash t : n]\!]$ in any action model.

2. (Completeness) Given derivations $\Gamma \vdash s : n$ and $\Gamma \vdash t : n$ in $\mathsf{T}(\mathbb{K})$, if $[\![\Gamma \vdash s : n]\!] = [\![\Gamma \vdash t : n]\!]$ in every action model then $\Gamma \vdash s = t : n$.

3. (Initiality) There is an initial term model $\mathcal{A}_0$ in $\mathbf{AMod}(\mathbb{K})$. □

The results above only deal with the static part of the type theory (hence action calculi). We can give a semantic interpretation of dynamics in an action model as a *local preorder* $\succeq$ on the symmetric monoidal category, such that the arrows coming from the cartesian category are minimal and satisfying the condition that $id_X \otimes f \succeq g$ implies $f \succeq g'$ for some g' and $g = id_X \otimes g'$. This may look a bit ad hoc, but is a natural interpretation of the condition of dynamics for action calculi (Definition 2.3).

Example 4.6 To interpret an action calculus with non-determinism primitive plus in Example 2.4, one may choose $\mathcal{C}$ as (the strict equivalent of) the category of sets and functions, and $\mathcal{S}$ as (the strict equivalent of) the category of sets and total relations; $\mathcal{F}$ is then the obvious inclusion functor. The interpretation of plus is simply given by the union of relations. We can then interpret the dynamics by the local preorder of the inclusion of relation. Obviously a total relation is minimal with respect to this preorder if and only if it is a function: that is, it comes from $\mathcal{C}$. A key feature of action calculi is that the dynamics are not usually closed under the control operators. Notice that in this model the preorder is closed under the natural transformation, so the model does not capture this feature. □

4.2 Higher-Order Extension

The intuition behind the definition of an action model with carrier $(\mathcal{C}, \mathcal{S}, \mathcal{F})$ is that the cartesian category $\mathcal{C}$ describes the behaviour of names in action calculi. In the higher-order case, the codes (λ-terms) have a similar behaviour to the names in that they can be substituted for names of the appropriate arity. Our definition of higher-order models extends action models to reflect this fact, by requiring that the functor $\mathcal{F}(_) \otimes X : \mathcal{C} \to \mathcal{S}$ has a right adjoint representing the arrow construction, which naturally extends the definition of exponents in cartesian closed categories given by right adjoint of the product functors $(_) \times X$.

Definition 4.7 A *higher-order action model* over signature $\mathbb{K}$ is an action model with carrier $(\mathcal{C}, \mathcal{S}, \mathcal{F})$ such that the functor $\mathcal{F}(_) \otimes X : \mathcal{C} \to \mathcal{S}$ has a (chosen) right adjoint $X \Rightarrow _ : \mathcal{S} \to \mathcal{C}$ for each object X, where the definition of $[\![m]\!]$ is adapted to the higher-order case by an additional requirement that $[\![m \Rightarrow n]\!] = [\![m]\!] \Rightarrow [\![n]\!]$. Again, a higher-order action model is *faithful* when the functor $\mathcal{F}$ is faithful, and it is *small* if the categories $\mathcal{C}$ and $\mathcal{S}$ are small. □

We write $\mathbf{ap} : (X \Rightarrow Y) \otimes X \to Y$ (in $\mathcal{S}$) for (components of) the counit of the adjunction, and $f^* : A \to X \Rightarrow B$ in $\mathcal{C}$ for the adjunct of $f : A \otimes X \to B$ in $\mathcal{S}$, i.e. f^* is the unique arrow satisfying $(\mathcal{F}(f^*) \otimes X); \mathbf{ap} = f$.

Definition 4.8 Let $\mathcal{H}_1$ with carrier $(\mathcal{C}, \mathcal{S}, \mathcal{F})$ and $\mathcal{H}_2$ with carrier $(\mathcal{C}', \mathcal{S}', \mathcal{F}')$ be higher-order action models over the same signature $\mathbb{K}$. A *higher-order action morphism* $f : \mathcal{H}_1 \to \mathcal{H}_2$ is an action morphism (f_c, f_s) such that it is a map of adjoints ([Mac71], p.97) from $\mathcal{F}(_) \otimes X \dashv X \Rightarrow (_)$ to $\mathcal{F}'(_) \otimes f_s(X) \dashv f_s(X) \Rightarrow' (_)$: that is, $f_c(X \Rightarrow (_)) = f_s(X) \Rightarrow' f_s(_)$ holds and the following diagram commutes for each X, A and B.

$$\begin{array}{ccc}
\mathcal{S}(\mathcal{F}(A)\otimes X, B) & \xrightarrow{(_)^*} & \mathcal{C}(A, X\Rightarrow B) \\
f_s(_)\downarrow & & \downarrow f_c(_) \\
\mathcal{S}'(f_s(\mathcal{F}(A))\otimes X), f_s(B)) & & \mathcal{C}'(f_c(A), f_c(X\Rightarrow B)) \\
\| & & \| \\
\mathcal{S}'(\mathcal{F}'(f_c(A))\otimes' f_s(X), f_s(B)) & \xrightarrow{(_)^*} & \mathcal{C}'(f_c(A), f_s(X)\Rightarrow' f_s(B))
\end{array}$$

□

The *category of small higher-order action models*, **HAMod**($\mathbb{K}$), is the category whose objects are the small higher-order action models, and whose morphisms are the higher-order action morphisms, with the obvious identities and composition.

The interpretation of the type theory $\mathsf{HT}(\mathcal{K})$ in an arbitrary higher-order action model $\mathcal{H}$ is given as an extension of Definition 4.4 to account for λ-abstraction and application.

Definition 4.9 Given type theory $\mathsf{HT}(\mathbb{K})$ and higher-order action model $\mathcal{H}$, the *interpretation* $[\![_]\!]$ of sequents $\Gamma \vdash t : m$ in the type theory as morphisms $[\![\Gamma \vdash t : m]\!] : [\![\Gamma]\!] \to [\![m]\!]$ in $\mathcal{S}$ is defined by induction on the structure of the derivation of sequents, as given in Definition 4.4 and the additional cases:

$$\begin{array}{lcl}
[\![\Gamma \vdash \lambda\vec{x}.t : m\Rightarrow n]\!] & = & \mathcal{F}([\![\Gamma, \vec{x}^m \vdash t : n]\!]^*) \\
[\![\Gamma \vdash st : n]\!] & = & \mathcal{F}(\Delta_{[\![\Gamma]\!]}); ([\![\Gamma \vdash s : m\Rightarrow n]\!]\otimes [\![\Gamma \vdash t : m]\!]); \mathbf{ap}
\end{array}$$

□

While soundness of the interpretation is routinely checked, the construction of the term model from the type theory (or the closed fragment of the higher-order action calculus) requires some careful calculation. In this case the cartesian category part is constructed from the values: that is, the variables and λ-abstractions. The non-trivial part is to check that $X\Rightarrow(_)$ does give a right adjoint of $\mathcal{F}(_)\otimes X$, which requires the axiom η_0 (η_{name} in higher-order action calculi). Without this axiom, $X\Rightarrow(_)$ fails to be a functor, though still it is a semifunctor, and in fact gives rise to a right semiadjoint [Hay85] of $\mathcal{F}(_)\otimes X$. In this sense, the semantic role of the axiom η_0 (η_{name}) is the same as that of the η axiom of the simply typed lambda calculus.

Theorem 4.10

1. (Soundness) Given $\Gamma \vdash s = t : n$ in $\mathsf{HT}(\mathbb{K})$, we have $[\![\Gamma \vdash s : n]\!] = [\![\Gamma \vdash t : n]\!]$ in any higher-order action model.
2. (Completeness) Given derivations $\Gamma \vdash s : n$ and $\Gamma \vdash t : n$ in $\mathsf{HT}(\mathbb{K})$, if $[\![\Gamma \vdash s : n]\!] = [\![\Gamma \vdash t : n]\!]$ in every higher-order action model then $\Gamma \vdash s = t : n$ in $\mathsf{HT}(\mathbb{K})$.
3. (Initiality) There is an initial term model $\mathcal{H}_0$ in **HAMod**($\mathbb{K}$). □

Remark 4.11 In a higher-order action model, it is easy to show using the Yoneda lemma that

$$\begin{array}{ll}
 & \mathbf{Set}^{\mathcal{C}^{\text{op}}}(\prod_{i=1,\ldots,r}\mathcal{S}(\mathcal{F}(_)\otimes A_i, B_i), \mathcal{S}(\mathcal{F}(_)\otimes A, B)) \\
\simeq & \mathcal{S}(\bigotimes_{i=1,\ldots r}\mathcal{F}(A_i\Rightarrow B_i)\otimes A, B) \\
\simeq & \mathcal{C}(\prod_{i=1,\ldots,r}(A_i\Rightarrow B_i), A\Rightarrow B)
\end{array}$$

This fact means that it is possible to interpret the control operators as morphisms rather than natural transformations in the higher-order setting. Syntactically, this means that we can replace parameterized controls by non-parameterized (higher-order) ones without changing the equational theory. We do not know how this affects the dynamics in general. □

Example 4.12 The semantic model of the calculus with non-determinism primitive plus, given in Example 4.6, is in fact a higher-order action model: let $X \Rightarrow Y$ be the set of total relations from X to Y, and for a total relation $R : A \otimes X \to B$ (the tensor product is just the direct product of sets) its adjunct $R^* : A \to X \Rightarrow B$ (in $\mathcal{C}$, hence a function) is determined by $xR^*(a)b$ if and only if $(a, x)Rb$. The above remark about the controls tells us that the interpretation of plus can be given as a total relation from $([\![m]\!] \Rightarrow [\![n]\!]) \otimes ([\![m]\!] \Rightarrow [\![n]\!]) \otimes [\![m]\!]$ to $[\![n]\!]$; explicitly it is given by the relation R where $(r, s, x)Ry$ if and only if $x(r \cup s)y$. □

4.3 Reflexive Extension

Now we turn to the models for the reflexive setting. For interpreting the reflexion or the letrec binding, we wish to have a semantic structure for representing these kind of circularity. Fortunately such a concept has been developed by Joyal, Street and Verity, called *traced monoidal categories* [JSV96]. Following the work by Mifsud [Mif96] and Hasegawa [Has97b], we show that traced monoidal categories can be used as models of the reflexive action calculi. Though the notion of trace is defined for general balanced monoidal categories (monoidal categories with braiding and twisting), for our purpose we only need the case of symmetric monoidal categories as presented in [Has97a, Has97b]. Further discussions and applications of traced monoidal categories in the context of computer science are found in [Abr96, ABP98, BCS98]. As a precursor we note that Căzănescu and Ştefănescu had the notion of *biflows* in mid '80s [CS88] whose axiomatization is equivalent to that of traced symmetric monoidal categories; see also [Mil94b] for historical remarks on related concepts.

Definition 4.13 A symmetric monoidal category $\mathcal{C}$ 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:**
$$Tr^X_{X,X}(c_{X,X}) = id_X : X \longrightarrow X$$
where c is the symmetry, i.e. $c_{X,Y} : X \otimes Y \longrightarrow Y \otimes X$.

□

We may omit the subscripts if there is no confusion. To help with the intuition, we present a graphical version of these axioms as well as three naturality conditions in Appendix A. We note that a symmetric monoidal category can be traced in many ways, so when we talk about a traced symmetric monoidal category we always fix a specific choice.

Now the models of the reflexive action calculi are simply defined by assuming that the symmetric monoidal category part is traced.

Definition 4.14 A *reflexive action model* over signature $\mathbb{K}$ is an action model with carrier $(\mathcal{C}, \mathcal{S}, \mathcal{F})$ such that the symmetric monoidal category $\mathcal{S}$ is *traced*. A *reflexive action morphism* between reflexive action models is an action morphism which preserves trace on the nose. □

We write $\mathbf{AMod}^r(\mathbb{K})$ for the category of small reflexive action models.

Definition 4.15 Given type theory $\mathsf{T}^r(\mathbb{K})$ and reflexive action model $\mathcal{R}$, the *interpretation* $[\![_]\!]$ of sequents $\Gamma \vdash t : m$ in the type theory as morphisms $[\![\Gamma \vdash t : m]\!] : [\![\Gamma]\!] \to [\![m]\!]$ in $\mathcal{S}$ is defined by induction on the structure of the derivation of sequents, as given in Definition 4.4 except the case of the letrec binding:

$$[\![\Gamma \vdash \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t]\!] = \mathcal{F}(\Delta_{[\![\Gamma]\!]}); (id_{[\![\Gamma]\!]} \otimes Tr^{[\![m]\!]}([\![\Gamma, \vec{x}^m \vdash s : m]\!]; \Delta_{[\![m]\!]})); [\![\Gamma, \vec{x}^m \vdash t : n]\!]$$

□

We give a graphical explanation of this interpretation in Appendix A.

It is routine to prove the soundness and completeness of the interpretation, which is similar to checking the correspondence between the reflexive action calculi and the type theory. A detailed calculation and discussion can be found in [Mif96, Has97b].

Theorem 4.16

1. (Soundness) Given $\Gamma \vdash s = t : n$ in $\mathsf{T}^r(\mathbb{K})$, we have $[\![\Gamma \vdash s : n]\!] = [\![\Gamma \vdash t : n]\!]$ in any reflexive action model.
2. (Completeness) Given derivations $\Gamma \vdash s : n$ and $\Gamma \vdash t : n$ in $\mathsf{T}^r(\mathbb{K})$, if $[\![\Gamma \vdash s : n]\!] = [\![\Gamma \vdash t : n]\!]$ in every reflexive action model then $\Gamma \vdash s = t : n$ in $\mathsf{T}^r(\mathbb{K})$.
3. (Initiality) There is an initial term model in $\mathbf{AMod}^r(\mathbb{K})$. □

Example 4.17 We slightly modify the semantic model of the calculus with non-determinism primitive plus, given in Example 4.6, to accommodate reflexion. As Example 4.6 we choose $\mathcal{C}$ as the category of sets and functions. For $\mathcal{S}$, we take the category of sets and (not necessarily total) relations, with $\mathcal{F}$ the obvious inclusion. Then the category $\mathcal{S}$ is traced in the following sense: given a relation $r : A \times X \to B \times X$, we define its trace $Tr^X(r) : A \to B$ by

$$a\ Tr^X(r)\ b \quad \text{iff} \quad \exists x \in X\ (a, x)\ r\ (b, x)$$

Thus this setting gives a sound semantic model of the reflexive action calculus with the non-determinism primitive plus. Actually this is at the same time a higher-order action model, thus we can interpret the higher-order reflexive action calculus in this setting (see Section 7). □

5 Conservativity of Higher-Order Extension

Since the higher-order theories (higher-order action calculi or our higher-order type theories) are obtained from the first-order ones by adding new constructs and additional axioms, there is an obvious sound translation from the first-order theories to the higher-order ones. A non-trivial fact is that this translation is (statically) *conservative*. In [Mil94a], Milner gives a syntactic proof of conservativity by appealing to a normal form result on higher-order action calculi, using the so-called higher-order molecular forms. It is easy to extend his syntactic result to incorporate our axiom η_{name}. Here we give another proof, using the properties of our semantic models. While conservativity of the reflexive action calculi over the action calculi is also true and syntactically provable using reflexive molecular forms [Mil94b], it seems that our semantic method cannot be used for the reflexive setting – see the discussion at the end of this section.

The conservativity result is achieved by constructing a higher-order action model into which the term model of the first-order theory faithfully embeds. In the case of standard algebraic type theory, the corresponding result is well-known. Given a cartesian category $\mathcal{C}$, the presheaf category $[\mathcal{C}^{\mathrm{op}}, \mathbf{Set}]$ is a cartesian closed category and the Yoneda embedding is fully faithful and preserves products. Applying this fact to the term model for a first-order algebraic theory, it follows that the algebraic type theory can be faithfully embedded in the corresponding higher-order type theory. However, additional care is required for our setting, since we need to deal with a symmetric monoidal category and a functor, as well as a cartesian category, and moreover our theory contains parameterized control operators which do not exist in standard algebraic theories.

We start with a known fact about the Yoneda construction (free cocompletion) on symmetric monoidal categories [Day70]. A systematic account can be found in [PR97].

Lemma 5.1 Let $\mathcal{C}$, $\mathcal{D}$ be small (strict) symmetric monoidal categories and $\mathcal{F} : \mathcal{C} \longrightarrow \mathcal{D}$ an identity-on-objects strict symmetric monoidal functor. There exists a small (strict) symmetric monoidal closed category $\bar{\mathcal{C}}$, a small (strict) symmetric monoidal category $\bar{\mathcal{D}}$ and fully faithful strict symmetric monoidal functors $in_{\mathcal{C}} : \mathcal{C} \longrightarrow \bar{\mathcal{C}}$ and $in_{\mathcal{D}} : \mathcal{D} \longrightarrow \bar{\mathcal{D}}$, together with an identity-on-objects strict symmetric monoidal functor $\bar{\mathcal{F}} : \bar{\mathcal{C}} \longrightarrow \bar{\mathcal{D}}$ such that $\bar{\mathcal{F}} \circ in_{\mathcal{C}} = in_{\mathcal{D}} \circ \mathcal{F}$ and $\bar{\mathcal{F}}$ has a right adjoint. Moreover $in_{\mathcal{C}}$ is dense.

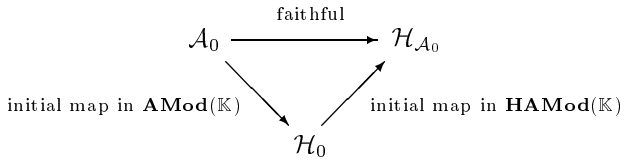
Proof: Let $\bar{\mathcal{C}}$ be the presheaf category $[\mathcal{C}^{\mathrm{op}}, \mathbf{Set}]$ and $in_{\mathcal{C}}$ be the Yoneda embedding (which is dense [Mac71]). It is well-known that $\bar{\mathcal{C}}$ is the free symmetric monoidal cocompletion of $\mathcal{C}$ and $in_{\mathcal{C}}$ is strict symmetric monoidal [Day70, IK86]. (If $\mathcal{C}$ is strict, let $\bar{\mathcal{C}}$ be the strict equivalent of $[\mathcal{C}^{\mathrm{op}}, \mathbf{Set}]$.) Then $\mathcal{F}$ extends to a strict symmetric monoidal functor $\hat{\mathcal{F}} : \bar{\mathcal{C}} \longrightarrow [\mathcal{D}^{\mathrm{op}}, \mathbf{Set}]$ with a right adjoint $U = [\mathcal{F}^{\mathrm{op}}, \mathbf{Set}]$, so that $\hat{\mathcal{F}}$ strictly commutes with $\mathcal{F}$. Although $\hat{\mathcal{F}}$ may not be the identity on objects, we can factorize it as $\hat{\mathcal{F}} = J \circ \bar{\mathcal{F}}$ so that $\bar{\mathcal{F}} : \bar{\mathcal{C}} \longrightarrow \bar{\mathcal{D}}$ is the identity on objects and $J : \bar{\mathcal{D}} \longrightarrow [\mathcal{D}^{\mathrm{op}}, \mathbf{Set}]$ is fully faithful. A right adjoint of $\bar{\mathcal{F}}$ is then given by $U \circ J : \bar{\mathcal{D}} \longrightarrow \bar{\mathcal{C}}$. The categories obtained above are not small, but we can cut down them to be small and still retain the required structure. □

Proposition 5.2 Given an action model $\mathcal{A}$ over signature $\mathbb{K}$, there is a higher-order action model $\mathcal{H}_{\mathcal{A}}$ over the same signature $\mathbb{K}$ such that there is a faithful action morphism from $\mathcal{A}$ to $\mathcal{H}_{\mathcal{A}}$.

Proof: We just apply the previous lemma. Let $(\mathcal{C}, \mathcal{S}, \mathcal{F})$ be the carrier of $\mathcal{A}$. Since $\bar{\mathcal{C}}$ is cartesian closed and $\bar{\mathcal{F}}$ has a right adjoint (say U), the functor $\bar{\mathcal{F}}(_) \otimes X$ also has a right adjoint $X \Rightarrow (_) = (U(_))^X$. We show that this carrier $(\bar{\mathcal{C}}, \bar{\mathcal{S}}, \bar{\mathcal{F}})$ gives a higher-order action model. We choose the interpretation of arities $[\![_]\!]'$ so that $[\![m]\!]' = in_{\mathcal{C}}([\![m]\!])$ for first-order arity m. Now we consider the interpretation of controls. For simplicity, we consider just

the case of single parameter. Assume that there is a family of functions $K_X : \mathcal{S}(\mathcal{F}(X) \otimes A, B) \longrightarrow \mathcal{S}(\mathcal{F}(X) \otimes C, D)$ (natural in X in $\mathcal{C}$). Since $in_{\mathcal{S}}$ is fully faithful strict symmetric monoidal and $\bar{\mathcal{F}} \circ in_{\mathcal{C}} = in_{\mathcal{S}} \circ \mathcal{F}$, this induces a family of functions $K'_X : \bar{\mathcal{S}}(\bar{\mathcal{F}}(in_{\mathcal{C}}(X)) \otimes in_{\mathcal{S}}(A), in_{\mathcal{S}}(B)) \longrightarrow \bar{\mathcal{S}}(\bar{\mathcal{F}}(in_{\mathcal{C}}(X)) \otimes in_{\mathcal{S}}(C), in_{\mathcal{S}}(D))$ natural in X in $\mathcal{C}$. Since $in_{\mathcal{C}}$ is dense, we can extend K' to a family of functions $K''_X : \bar{\mathcal{S}}(\bar{\mathcal{F}}(X) \otimes in_{\mathcal{S}}(A), in_{\mathcal{S}}(B)) \longrightarrow \bar{\mathcal{S}}(\bar{\mathcal{F}}(X) \otimes in_{\mathcal{S}}(C), in_{\mathcal{S}}(D))$ natural in X in $\bar{\mathcal{C}}$. Thus given an interpretation of a control in $\mathcal{A}$, we have an interpretation of the control in $\mathcal{H}_{\mathcal{A}}$ of the carrier $(\bar{\mathcal{C}}, \bar{\mathcal{S}}, \bar{\mathcal{F}})$. And $(in_{\mathcal{C}}, in_{\mathcal{S}})$ is a faithful action morphism. □

Taking $\mathcal{A}$ as the term model $\mathcal{A}_0$, we obtain a higher-order action model $\mathcal{H}_{\mathcal{A}_0}$ into which $\mathcal{A}_0$ faithfully embeds. Because $\mathcal{H}_0$ is initial in **HAMod**($\mathbb{K}$), there is a unique higher-order action morphism from $\mathcal{H}_0$ to $\mathcal{H}_{\mathcal{A}_0}$. On the other hand, since $\mathcal{A}_0$ is initial in **AMod**($\mathbb{K}$) and also **HAMod**($\mathbb{K}$) is a subcategory of **AMod**($\mathbb{K}$), the following diagram commutes in **AMod**($\mathbb{K}$):



Therefore the unique action morphism from $\mathcal{A}_0$ to $\mathcal{H}_0$ must be faithful.

Theorem 5.3 The higher-order theory $\mathsf{HT}(\mathbb{K})$ is a conservative extension of the first-order theory $\mathsf{T}(\mathbb{K})$. □

Remark 5.4 We note that a stronger result called *full completeness* holds. Full completeness states that, in addition to conservativity, for any term $\Gamma \vdash t : m$ of $\mathsf{HT}(\mathbb{K})$ with no higher-order arities in Γ nor m, there exists a term $\Gamma \vdash s : m$ of $\mathsf{T}(\mathbb{K})$ such that $\Gamma \vdash s = t : m$ is probable in $\mathsf{HT}(\mathbb{K})$. This result is syntactically derivable just by inspecting Milner's conservativity proof using molecular forms [Mil94a] though Milner did not state it explicitly. In the semantic setting, full completeness amounts to saying that the corresponding action morphism between the term models is fully faithful. It is possible to modify our proof method for conservativity to that for full completeness by constructing yet another model using categorical gluing (see for instance [Cro93, MS93]). However this semantic proof involves several category theoretic tools which are behind the scope of this paper, and we postpone it to a forthcoming paper. □

It seems difficult to scale up our proof method to the reflexive setting, essentially because our model construction does not respect trace. We also note that conservativity of the higher-order reflexive theory (see Section 8) over the first-order reflexive theory remains open. Our semantic proof method does not help. Syntactically the higher-order reflexive theory is neither confluent (by the same reason as the non-confluence of cyclic lambda calculi [AK94]) nor normalizing. It therefore seems hard to extend Milner's proof for the reflexive setting.

6 Notions of Computation

We show that the higher-order action calculi conservatively extend Moggi's computational λ-calculus [Mog88], with an additional axiom for commuting let-bindings. Whilst it is possible to give a syntactic proof of this result, we give a simpler comparison by showing that the

models of higher-order action calculi correspond to Moggi's semantic framework called *notions of computation* [Mog91].

Independently, Pavlovic has also mentioned the connection between his models of action calculi and notions of computation [Pav97].

6.1 The Computational Lambda Calculus

We reproduce the computational lambda calculus (λ_c-calculus) below, using the simply typed version as found in [MOTW95].

Definition 6.1 The *computational lambda calculus* is determined by the following data.

1. [Types] The set of *types* over the set P (called base types) is defined by the grammar

$$\sigma ::= p \in \mathsf{P} \mid \sigma \Rightarrow \sigma$$

2. [Terms and Values] The sets of *terms* and *values* over the set of variables V is defined by the grammar

$$\begin{array}{lllll} \text{terms} & t & ::= & x \in V \mid \lambda x.t \mid tt \mid \mathsf{let}\ x\ \mathsf{be}\ t\ \mathsf{in}\ t \\ \text{values} & v & ::= & x \mid \lambda x.t \end{array}$$

3. [Typing judgements] We say that a term is well-typed if it can be shown to annotate a sequent using the rules

$$\frac{}{\Gamma, x^\sigma \vdash_c x : \sigma} \qquad \frac{\Gamma, x^\sigma \vdash_c t : \tau}{\Gamma \vdash_c \lambda x.t : \sigma \Rightarrow \tau}$$

$$\frac{\Gamma \vdash_c s : \sigma \Rightarrow \tau \quad \Gamma \vdash_c t : \sigma}{\Gamma \vdash_c st : \tau} \qquad \frac{\Gamma \vdash_c s : \sigma \quad \Gamma, x^\sigma \vdash_c t : \tau}{\Gamma \vdash_c \mathsf{let}\ x\ \mathsf{be}\ s\ \mathsf{in}\ t : \tau}$$

$$\frac{\Gamma, x^\sigma, y^{\sigma'}, \Gamma' \vdash_c t : \tau}{\Gamma, y^{\sigma'}, x^\sigma, \Gamma' \vdash_c t : \tau}$$

4. [Equality judgements] We define an equality judgement $\Gamma \vdash_c s = t : \sigma$, where $\Gamma \vdash_c s : \sigma$ and $\Gamma \vdash_c t : \sigma$, as the congruence relation generated by the axioms:

(β_v)	$(\lambda x.t)v$	$=$	$t\{v/x\}$
(η_v)	$\lambda x.vx$	$=$	v, for x not free in v
(let_v)	$\mathsf{let}\ x\ \mathsf{be}\ v\ \mathsf{in}\ t$	$=$	$t\{v/x\}$
(id)	$\mathsf{let}\ x\ \mathsf{be}\ t\ \mathsf{in}\ x$	$=$	t
(comp)	$\mathsf{let}\ y\ \mathsf{be}\ (\mathsf{let}\ x\ \mathsf{be}\ s\ \mathsf{in}\ t)\ \mathsf{in}\ u$	$=$	$\mathsf{let}\ x\ \mathsf{be}\ s\ \mathsf{in}\ \mathsf{let}\ y\ \mathsf{be}\ t\ \mathsf{in}\ u$
(let.1)	et	$=$	$\mathsf{let}\ z\ \mathsf{be}\ e\ \mathsf{in}\ zt$
(let.2)	ve	$=$	$\mathsf{let}\ x\ \mathsf{be}\ e\ \mathsf{in}\ vx$

where e ranges over non-values: that is, applications and let-blocks.

□

It is not hard to see that the obvious translation from the computational lambda calculus (with the base types P) to our higher-order type theory $\mathsf{HT}(\mathbb{K})$ (with $\mathbb{K} = (\mathsf{P}, \mathcal{K})$) is sound. However, this translation is not conservative. To achieve conservativity, we need to strengthen the equational theory of the computational lambda calculus, by assuming an additional axiom for *commuting* let-bindings (which is derivable in our theory, see Lemma 3.5).

Definition 6.2 The *commutative* computational lambda calculus is obtained by adding the following axiom:

$$\text{(comm)} \quad \mathsf{let}\ x\ \mathsf{be}\ s\ \mathsf{in}\ \mathsf{let}\ y\ \mathsf{be}\ t\ \mathsf{in}\ u\ =\ \mathsf{let}\ y\ \mathsf{be}\ t\ \mathsf{in}\ \mathsf{let}\ x\ \mathsf{be}\ s\ \mathsf{in}\ u$$

(As in other axioms, both sides of the axiom must have the same type under the same context, therefore x cannot be free in t and y cannot be free in s.) □

Theorem 6.3 The higher-order type theory $\mathsf{HT}(\mathbb{K})$ is a conservative extension of the commutative computational lambda calculus. □

This theorem can be proved syntactically by comparing the normal forms in these calculi. Although there seems to be no simple confluent terminating rewriting system for the commutative computational λ-calculus, it is possible to define a notion of normal form which closely resembles Milner's higher-order molecular forms [Mil94a]. Rather than presenting this syntactic proof, we give a semantic proof of conservativity by relating the models.

6.2 Comparison with λ_c-Models

We show the connection between faithful higher-order action models and Moggi's models of the computational lambda calculus, called λ_c-*models* [Mog88]. This observation relies on fairly standard category theory, and is a special instance of the results given in [PR97]. To recall from Moggi's work, a monad (T, η, μ) satisfies the *mono requirement* if each component of η is a monomorphism. A *tensorial strength* for a monad T on a symmetric monoidal category is a natural transformation with components $\theta_{A,B} : A \otimes TB \to T(A \otimes B)$ subject to the coherence conditions as found in [Koc70, Mog88, Mog91]. It is *commutative* if the evident two natural transformations from $TA \otimes TB$ to $T(A \otimes B)$ agree.

Definition 6.4 A λ_c-*model* is a cartesian category $\mathcal{C}$ with a strong monad (T, η, μ) which satisfies the mono requirement and has *Kleisli exponents*: that is, for each object A, the functor $\mathcal{J}(A \times _) : \mathcal{C} \to \mathcal{C}_T$ has a (chosen) right adjoint $A \Rightarrow (_)$, where $\mathcal{C}_T$ is the Kleisli category of T and $\mathcal{J} : \mathcal{C} \to \mathcal{C}_T$ is given by $\mathcal{J}(f) = f; \eta$. A λ_c-model is said to be *commutative* if the tensorial strength is commutative. □

Proposition 6.5 To give a carrier $(\mathcal{C}, \mathcal{S}, \mathcal{F})$ such that $\mathcal{F}$ is faithful and $\mathcal{F}(_) \otimes X$ has a right adjoint for each X is to give a commutative λ_c-model in which cartesian products are strictly associative. □

Sketch of Proof: Given such a carrier $(\mathcal{C}, \mathcal{S}, \mathcal{F})$, we have a commutative strong monad on $\mathcal{C}$ as the composition of $\mathcal{F}$ with its right adjoint, which satisfies the mono requirement and has Kleisli exponents. Conversely, given a commutative λ_c-model T over a cartesian category $\mathcal{C}$, we have a carrier $(\mathcal{C}, \mathcal{C}_T, \mathcal{J})$ which satisfies the properties above. Moreover these constructions are inverse to each other. (See [PR97] for a detailed account.) □

Together with the observation in Remark 4.11, we can specify a higher-order action model as a commutative λ_c-model with the interpretation of primes and controls.

Theorem 6.6 To give a faithful higher-order action model over the signature $\mathbb{K} = (\mathsf{P}, \mathcal{K})$ is to give a commutative λ_c-model in which products are strictly associative, an object $[\![p]\!]_{\mathsf{P}}$ for each $p \in \mathsf{P}$ and an arrow

$$[\![\mathsf{K}]\!]_{\mathcal{K}} : \prod_{i=1,\ldots,r} ([\![m_i]\!] \Rightarrow [\![n_i]\!]) \to [\![m]\!] \Rightarrow [\![n]\!]$$

for each control K with arity rule $((m_1, n_1), \ldots, (m_r, n_r)) \to (m, n)$ where $[\![m]\!]$ is defined inductively as in the higher-order action models. □

Our result shows that the models of a *static* higher-order action calculus correspond to commutative λ_c-models, with interpretation functions for the primes and controls. To interpret the dynamics, we require an additional preorder enrichment of the Kleisli category, which is not present in Moggi's work.

Example 6.7 The model of plus (Example 2.4) given in Example 4.6 can be understood as a commutative λ_c-model: the corresponding monad is the non-empty powerset monad on the category of sets and functions. The interpretation $[\![\mathsf{plus}]\!]$ is given routinely, again by the union of relations. The local preorder on the Kleisli category (the category of sets and total relations) is derived from the subset inclusion on powersets. □

This example, used repeatedly in this paper, gives an elementary account of non-determinism. We can construct more sophisticated models of non-determinism along this line using various powerdomain monads. (In particular, the connection with models of the π-calculus studied in [FMS96, Sta96] requires further investigation.)

7 Reflexion and Recursion

Mifsud has observed how recursive computation can be created by combining higher-order and reflexive features [Mif96]. Later Hasegawa has shown that Mifsud's construction is possible in a very general context and formalized a new class of models of recursive computation in terms of traced monoidal categories [Has97a, Has97b]. Here we review Mifsud's construction using our type theory and show how this leads to Hasegawa's models of recursion.

7.1 Recursion in Higher-Order Reflexive Action Calculi

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 a recursive letrec binding, they can be written as $(\mathsf{letrec}\ x\ \mathsf{be}\ \lambda y.M(xy)\ \mathsf{in}\ x)$ and $(\mathsf{letrec}\ x\ \mathsf{be}\ \lambda y.x(My)\ \mathsf{in}\ x)$ respectively. We shall demonstrate these simple forms of iteration exist in the higher-order reflexive action calculus.

Let $a : m \to m$ be an action in the higher-order reflexive action calculus, which is the direct combination of the higher-order calculus with the reflexive one. In [Mif96] Mifsud defines actions ITER(a) and BACKITER(a) by

$$\frac{a : m \to m \quad x : n \Rightarrow m \quad x \notin fn(a)}{\mathrm{ITER}(a) \equiv (\mathrm{rec}((x)^{\ulcorner}(\langle x\rangle \otimes \mathbf{id}) \cdot \mathbf{ap} \cdot a^{\urcorner}) \otimes \mathbf{id}) \cdot \mathbf{ap} : n \to m}$$

$$\frac{a : m \to m \quad x : m \Rightarrow n \quad x \notin fn(a)}{\mathrm{BACKITER}(a) \equiv (\mathrm{rec}((x)^{\ulcorner}(\langle x\rangle \otimes a) \cdot \mathbf{ap}^{\urcorner}) \otimes \mathbf{id}) \cdot \mathbf{ap} : m \to n}$$

where $\mathrm{rec}_p(a)$ is given by

$$\frac{a : p \otimes m \to p \otimes n}{\mathrm{rec}_p(a) \equiv \uparrow_p (a \cdot (\Delta_p \otimes \mathbf{id}_n)) : m \to p \otimes n}$$

They satisfy the equations below.

$$\mathrm{ITER}(a) \;=\; \mathrm{ITER}(a) \cdot a$$

$$\mathrm{BACKITER}(a) \;=\; a \cdot \mathrm{BACKITER}(a)$$

The direct proof of these equations using the equaltional theory of the higher-order reflexive action calculi requires some long calculations [Mif96]. However, as we demonstrate below, they are easily and intuitively verified in our type theory for the higher-order reflexive action calculi. We present the syntax and axioms of the type theory in Appendix B. Assume that we have a closed term $M : m \Rightarrow m$ of the type theory which satisfies $\Psi(\ \vdash M) = \ulcorner a \urcorner$ where $\Psi(-)$ indicates the translation from the type theory to the higher-order reflexive action calculus. Then we have

$$\mathrm{ITER}(a) \quad = \quad \Psi(\vec{z}^{n} \vdash \mathsf{letrec}\ x\ \mathsf{be}\ \lambda\vec{y}.M(x\vec{y})\ \mathsf{in}\ x\vec{z} : m)$$

$$\mathrm{BACKITER}(a) \quad = \quad \Psi(\vec{z}^{m} \vdash \mathsf{letrec}\ x\ \mathsf{be}\ \lambda\vec{y}.x(M\vec{y})\ \mathsf{in}\ x\vec{z} : n)$$

For instance, one can prove $\mathrm{ITER}(a) = \mathrm{ITER}(a) \cdot a$ as follows. In the type theory, we have

$$\begin{array}{ll} & \mathsf{letrec}\ x\ \mathsf{be}\ \lambda\vec{y}.M(x\vec{y})\ \mathsf{in}\ x\vec{z} \\ = & \mathsf{letrec}\ x\ \mathsf{be}\ \lambda\vec{y}.M(x\vec{y})\ \mathsf{in}\ (\lambda(\vec{y}).M(x\vec{y}))\vec{z} \\ = & \mathsf{letrec}\ x\ \mathsf{be}\ \lambda\vec{y}.M(x\vec{y})\ \mathsf{in}\ M(x\vec{z}) \\ = & M(\mathsf{letrec}\ x\ \mathsf{be}\ \lambda\vec{y}.M(x\vec{y})\ \mathsf{in}\ x\vec{z}) \end{array}$$

hence

$$\begin{array}{ll} & \mathrm{ITER}(a) \\ = & \Psi(\vec{z} : n \vdash \mathsf{letrec}\ x\ \mathsf{be}\ \lambda\vec{y}.M(x\vec{y})\ \mathsf{in}\ x\vec{z} : m) \\ = & \Psi(\vec{z} : n \vdash M(\mathsf{letrec}\ x\ \mathsf{be}\ \lambda\vec{y}.M(x\vec{y})\ \mathsf{in}\ x\vec{z}) : m) \\ = & (\Psi(\vdash M : m \Rightarrow m) \otimes \Psi(\vec{z} : n \vdash \mathsf{letrec}\ x\ \mathsf{be}\ \lambda\vec{y}.M(x\vec{y})\ \mathsf{in}\ x\vec{z} : m)) \cdot \mathbf{ap} \\ = & \Psi(\vec{z} : n \vdash \mathsf{letrec}\ x\ \mathsf{be}\ \lambda\vec{y}.M(x\vec{y})\ \mathsf{in}\ x\vec{z} : m) \cdot a \\ = & \mathrm{ITER}(a) \cdot a. \end{array}$$

7.2 Recursion and Trace

The observation above suggests that such a construction of fixpoints is always possible in any models of higher-order reflexive action calculi. Hasegawa has shown fixed point theorems for such semantic structures. We conclude this section by recalling his results and putting Mifsud's observation in this general context.

Theorem 7.1 [Has97a, Has97b] Let $\mathcal{F} : \mathcal{C} \longrightarrow \mathcal{S}$ be a strict symmetric monoidal functor from a cartesian category $\mathcal{C}$ to a traced symmetric monoidal category $\mathcal{S}$, which is the identity on objects and has a right adjoint. Then 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 [Has97b].) 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 $(-)^* : \mathcal{S}(A, B) \stackrel{\sim}{\longrightarrow} \mathcal{C}(A, UB)$. 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 $\mathcal{S}$. $\Box$

In particular this theorem applies to our models of higher-order reflexive action calculi. It is easy to derive ITER and BACKITER from this formula as special instances. In [Has97a] Hasegawa studies these categorical structures as a possible framework for recursive computation, with connection to cyclic lambda calculi.

Example 7.2 We have given a higher-order reflexive action model in Example 4.17. Applying the theorem, we obtain a fixpoint operator in this setting – explicitly we have a relation $r^\dagger : A \to X$ given by

$$a \; r^\dagger \; x \;\; \text{iff} \;\; \exists S \subseteq X \;\; S = \{y \mid \exists z \in S \;\; (a, z) \; r \; y\} \;\&\; x \in S$$

for a relation $r : A \times X \longrightarrow X$. $\Box$

Remark 7.3 Another proof of the theorem is possible using the type theory as an internal language. In the type theory this fixpoint construction amounts to

$$\frac{\Gamma, \vec{x}^m \vdash t : m}{\Gamma \vdash \mathsf{letrec}\ f^{\epsilon \Rightarrow m}\ \mathsf{be}\ \lambda.((\lambda \vec{x}.t)(f0))\ \mathsf{in}\ f0 : m}$$

and the properties stated in the theorem are derived using the axioms of the type theory, in the same manner as the calculation for ITER. $\Box$

On the other hand, a more specialized theorem on traced cartesian categories in [Has97a], proved independently by Hyland, states that trace and fixpoints are more or less equivalent in cartesian categories. For instance any models of domain theory are shown to be traced, in most cases trace is given by the least fixed point operators.

As a conceptual consequence, we see that recursion described in terms of traced categories is strictly more general than the traditional one. In particular the recursive computation in higher-order reflexive action calculi does not have to fit in the traditional models of recursion. The same seems to be also true for recursion created from cyclic graph rewriting system. We think that this opens up a new research direction which leads to a deeper understanding of recursive computation – see [BCS98] for a recent development in this direction.

8 Conclusions and Related Work

In this paper we have given the type theories for action calculi and their higher-order and reflexive extensions, and the categorical models of the extensions based on Power's models of action calculi. We have presented a semantic proof of the conservativity of (static) higher-order action calculi over action calculi, and have related our higher-order models with Moggi's λ_c-models. We also analysed recursive computation in higher-order reflexive action calculi, using the type-theoretic and categorical presentations given here.

Our results form part of a broader picture. First, there is work by Barber, Gardner, Hasegawa and Plotkin [BGHP96, BGHP98], which links action calculi with *intuitionistic linear type theory* [BP98, Ben95], and proves conservativity results for various extensions of action calculi incorporating the results given here. This work focusses more extensively on relating type theories by comparing their categorical models. (See also [BW96] for a related work on linear type theory and notions of computation.)

Another perspective is the connection with graph rewriting systems. The above mentioned work connecting action calculi and linear logic [BGHP98] should lead to an integration of

techniques for reasoning about the graphical presentation of actions (action graphs [Mil96]), proof nets and interaction nets. Hasegawa's thesis [Has97b] studies the category theoretic framework in this paper more extensively as the semantic models of *term graphs* arising from graph rewriting theory. Independently, similar models of term graphs called *gs-monoidal categories* have been proposed by Corradini and Gadducci [CG97]; Miyoshi has adapted Hasegawa's work for gs-monoidal categories and rewriting logic [Miy98]. It is easy to see that their models are very close to the action models, expect that they use **Cat**-enrichment for modelling rewriting whereas we use preorder-enrichment for dynamics. Action calculi fit well within the general theory of graph rewriting systems, and we believe that this direction should lead to a fruitful interaction between graph rewriting theory and concurrency theory.

Finally, whilst in this paper symmetric monoidal categories are sufficient for describing models of action calculi, Power and Robinson focus on *premonoidal categories* [PR97] for expressing computational behaviour, in which the tensor product need not be bifunctorial. They argue that this weaker structure is more natural for describing computational behaviour such as imperative features. Recently Jeffrey introduced a graphical view of imperative programs along this line [Jef98]. In essence, Jeffrey (and implicitly Power and Robinson) emphasises the notion of a control line through the graphs to specify the causal order of programs. This causal information can be expressed using the controls of action calculi, and further work is required to fully understand the comparisons between these approaches.

Acknowledgements

We enjoyed many helpful discussions with Andrew Barber, Gordon Plotkin on type theory, Alex Mifsud on reflexive action calculi and John Power on category theory.

References

[Abr96] Abramsky, S. (1996), Retracing some paths in process algebra, *in* "Proceedings, Concurrency Theory (CONCUR'96)", Lecture Notes in Computer Science, Vol. 1119, pp. 1–17, Springer-Verlag, Berlin/New York.

[ABP98] Abramsky, S., Blute, R., and Panangaden, P. (1998), Nuclear and trace ideals in tensored *-categories, Manuscript.

[Acz80] Aczel, P. (1980), Frege structures and the notions of proposition, truth and set, *in* "The Kleene Symposium", pp.31–59, North-Holland.

[AA95] Ariola, Z. M., and Arvind (1995), Properties of a first-order functional language with sharing, *Theoret. Comp. Sci.* **146**, 69–108.

[AK94] Ariola, Z.M., and Klop, J.W. (1994), Cyclic lambda graph rewriting, *in* "Proceedings, 9th Symposium on Logic in Computer Science (LICS'94)", pp. 416–425.

[BGHP96] Barber, A., Gardner, P., Hasegawa, M., and Plotkin, G. (1996), Action calculi, the computational λ-calculus and linear logic, Draft.

[BGHP98] Barber, A., Gardner, P., Hasegawa, M., and Plotkin, G. (1998), From action calculi to linear logic, *in* "Proceedings, Computer Science Logic (CSL'97)", Lecture Notes in Computer Science, Springer-Verlag, Berlin/New York.

[BP98] Barber, A., and Plotkin, G. (1998), Dual intuitionistic linear logic, Submitted.

[Ben95] Benton, N. (1995), A mixed linear non-linear logic: proofs, terms and models, *in* "Proceedings, Computer Science Logic (CSL'94)", Lecture Notes in Computer Science, Vol. 933, pp. 121–135, Springer-Verlag, Berlin/New York.

[BW96] Benton, N., and Wadler, P. (1996), Linear logic, monads, and the lambda calculus, *in* "Proceedings, 11th Symposium on Logic in Computer Science (LICS'96)".

[BCS98] Blute, R.F., Cockett, J.R.B., and Seely, R.A.G. (1998), Feedback for linearly distributive categories: traces and fixpoints, Manuscript.

[CS88] Căzănescu, V.-E., and Ştefănescu, Gh. (1988), A formal representation of flowchart schemes. *An. Univ. Bucureşti Mat. - Inf.* **2**, 33–51.

[CG97] Corradini, A., and Gadducci, F. (1997), A 2-categorical presentation of term graphs, *in* "Proceedings, Category Theory and Computer Science (CTCS'97)", Lecture Notes in Computer Science, Vol. 1290, pp. 87–105, Springer-Verlag, Berlin/New York.

[Cro93] Crole, R. (1993), "Categories for Types", Cambridge University Press.

[Day70] Day, B. J. (1970), On closed categories of functors. *in* "Midwest Category Seminar Reports IV", Lecture Notes in Mathematics, Vol. 137, pp. 1–38, Springer-Verlag, Berlin/New York.

[FMS96] Fiore, M. P., Moggi, E., and Sangiorgi, D. (1996), A fully-abstract model for the π-calculus, *in* "Proceedings, 11th Symposium on Logic in Computer Science (LICS'96)".

[Gar95] Gardner, P. (1995), A name-free account of action calculi, *in* "Proceedings, 11th International Conference on Mathematical Foundations of Programming Semantics (MFPS'95)", Electronic Notes in Computer Science **1**, Elsevier.

[Gar98] Gardner, P. (1998), Closed action calculi, *to appear in Theoret. Comput. Sci.*

[GH97] Gardner, P., and Hasegawa, M. (1997), Types and models for higher-order action calculi, *in* "Proceedings, Theoretical Aspects of Computer Software (TACS'97)", Lecture Notes in Computer Science , Vol. 1281, pp. 583–603, Springer-Verlag, Berlin/New York.

[Has97a] Hasegawa, M. (1997), Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi, *in* "Proceedings, Typed Lambda Calculi and Applications (TLCA'97)", Lecture Notes in Computer Science, Vol. 1210, pp. 196–213, Springer-Verlag, Berlin/New York.

[Has97b] Hasegawa, M. (1997), "Models of Sharing Graphs (A Categorical Semantics of Let and Letrec)", Ph.D. thesis, ECS-LFCS-97-360, University of Edinburgh.

[Hay85] Hayashi, S. (1985), Adjunction of semi-functors: categorical structures in non-extensional lambda-calculus, *Theoret. Comp. Sci.* **41**, 95–104.

[IK86] Im, G. B., and Kelly, G. M. (1986), A universal property of the convolution monoidal structure, *J. Pure Appl. Algebra* **43**, 75–88.

[Jef98] Jeffrey, A. (1998), Premonoidal categories and a graphical view of programs, Manuscript.

[JS91] Joyal, A., and Street, R. (1991), The geometry of tensor calculus I, *Adv. Math.* **88**, 55–113.

[JSV96] Joyal, A., Street, R., and Verity, D. (1996), Traced monoidal categories, *Math. Proc. Cambridge Philos. Soc.* **119** (3), 447–468.

[Koc70] Kock, A. (1970), Strong functors and monoidal monads, Various Publications Series 11, Aarhus Universitet.

[Mac71] Mac Lane, S. (1971), "Categories for the Working Mathematician", Graduate Texts in Mathematics 5, Springer-Verlag, Berlin/New York.

[MOTW95] Maraist, J., Odersky, M., Turner, D., and Wadler, P. (1995), Call-by-name, call-by-value, call-by-need, and the linear lambda calculus, *in* "Proceedings, 11th International Conference on Mathematical Foundations of Programming Semantics (MFPS'95)", Electronic Notes in Computer Science **1**, Elsevier.

[Mif96] Mifsud, A. (1996), "Control Structures", Ph.D. thesis, University of Edinburgh.

[MMP95] Mifsud, A., Milner, R., and Power, A. J. (1995), Control structures, *in* "Proceedings, 10th Symposium on Logic in Computer Science (LICS'95)".

[Mil94a] Milner, R. (1994), Higher-order action calculi, *in* "Proceedings, Computer Science Logic (CSL'93)", Lecture Notes in Computer Science, Vol. 832, pp. 238–260, Springer-Verlag, Berlin/New York.

[Mil94b] Milner, R. (1994), Action calculi V: reflexive molecular forms (with Appendix by O. Jensen), Unpublished manuscript.

[Mil96] Milner, R. (1996), Calculi for interaction, *Acta Inform.* **33**(8), 707–737.

[MS93] Mitchell, J.C., and Scedrov, A. (1992), Notes on sconing and relators, *in* "Computer Science Logic (CSL'92), Selected Papers", Lecture Notes in Computer Science, Vol. 702, pp. 352–378, Springer-Verlag, Berlin/New York.

[Miy98] Miyoshi, H. (1998), Rewriting logic for cyclic sharing structures, *in* "Proceedings, Third Fuji International Symposium on Functional and Logic Programming (FLOPS'98)", World Scientific, pp. 167–186.

[Mog88] Moggi, E. (1988), Computational lambda-calculus and monads, Technical report ECS-LFCS-88-66, University of Edinburgh.

[Mog91] Moggi, E. (1991), Notions of computation and monads. *Inform. and Comput.* **93**, 55–92.

[Pav97] Pavlović, D. (1997), Categorical logic of names and abstraction in action calculi, *Math. Structures Comp. Sci.* **7**(6), 619–637.

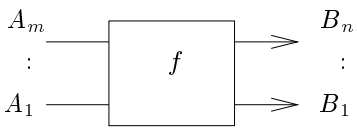
[Pow96] Power, A. J. (1996), Elementary control structures, *in* "Proceedings, Concurrency Theory (CONCUR'96)", Lecture Notes in Computer Science, Vol. 1119, pp. 115–130, Springer-Verlag, Berlin/New York.

[PR97] Power, A. J., and Robinson, E. P. (1997), Premonoidal categories and notions of computation, *Math. Structures Comp. Sci.* **7**(5), 453–468.

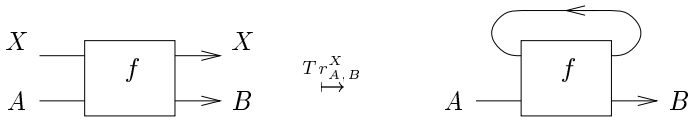
[Sta96] Stark, I. (1996), A fully abstract domain model for the π-calculus, *in* "Proceedings, 11th Symposium on Logic in Computer Science (LICS'96)", pp. 36–42.

A Graphical Presentation of Axioms for Trace

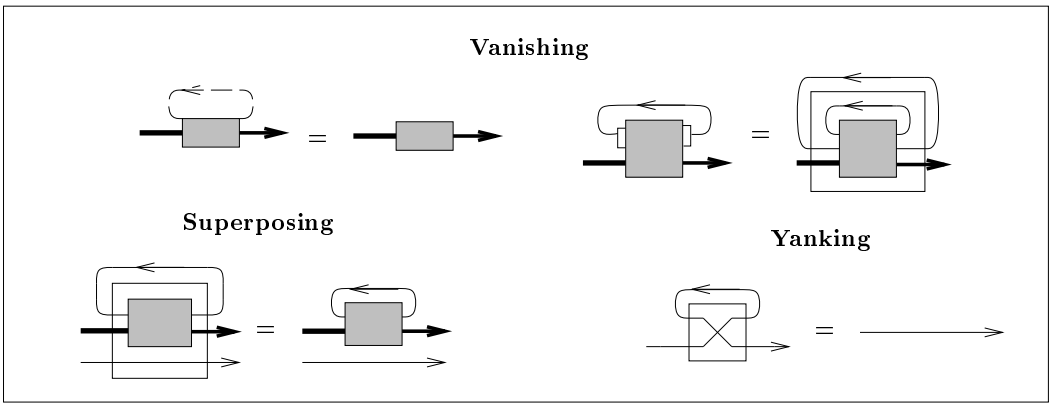
We present the graphical version of the axioms for traced symmetric monoidal categories, to help with the intuition of traced categories as categories with cycles (or feedback, reflexion). Such graphical languages for various monoidal categories have been developed in [JS91]. For an arrow $f : A_1 \otimes \ldots \otimes A_m \to B_1 \otimes \ldots \otimes B_n$ we draw a graph with m inputs and n outputs as



In particular the trace operator is nicely presented as an operator creating a cyclic binding:



Using this notation, the axioms for trace are given as follows.



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**)

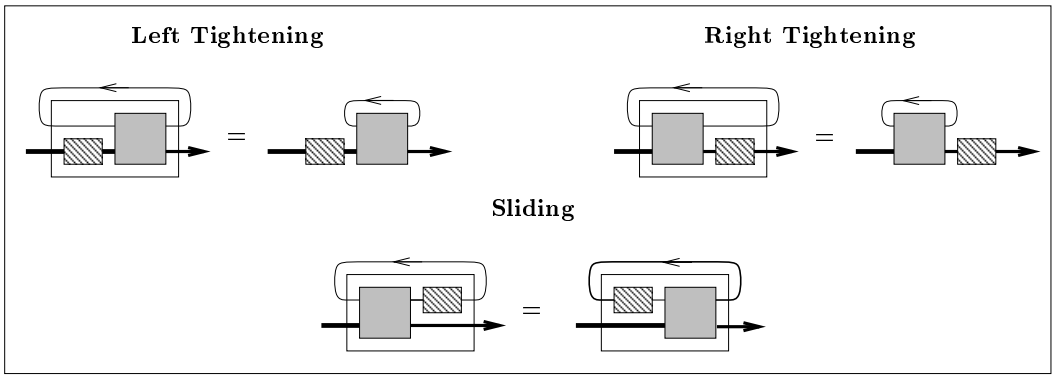
$$Tr^X_{A,B}(f; (g \otimes id_X)) = Tr^X_{A,B'}(f); g : A \longrightarrow B$$

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

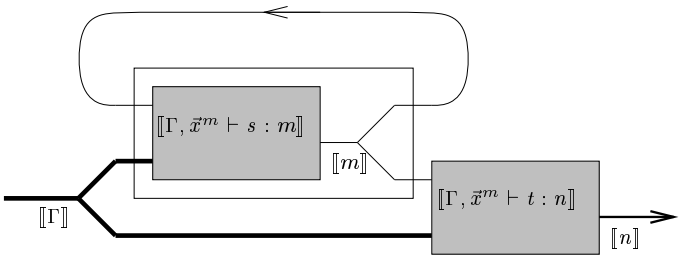


Remark A.1 The axiom **Superposing** is slightly simplified from the original version in [JSV96]

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

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

Example A.2 Using the graphical presentation, the semantic interpretation of the letrec binding in Definition 4.15 is given as



where the diagonal $\mathcal{F}(\Delta)$ is presented as a branching. □

B Higher-Order Reflexive Theory

The syntax and axiomatization of the type theory for higher-order action calculi is just the combination of the higher-order type theory with the reflexive theory. All definitions are summarised as below.

Definition B.1 The set of *type-theoretic terms* (usually just called *terms*) over $\mathbb{K}$ is defined by the following grammar:

$$t \ ::= \ x \mid 0 \mid tt \mid t \otimes t \mid \lambda\vec{x}.t \mid \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ t \mid \mathsf{K}((\vec{x})t, \ldots, (\vec{x})t; t)$$

where we assume the strict associativity (see Definition 3.1). Also we define *values* by

$$v \ ::= \ x \mid 0 \mid v \otimes v \mid \lambda\vec{x}.t$$

□

Definition B.2 A term is *well-typed* if it can be shown to annotate a sequent using the following rules.

$$\Gamma, x^p \vdash x : p \qquad \Gamma \vdash 0 : \epsilon \qquad \frac{\Gamma \vdash s : m \quad \Gamma \vdash t : n}{\Gamma \vdash s \otimes t : m \otimes n} \qquad \frac{\Gamma, \vec{x}^m \vdash s : m \quad \Gamma, \vec{x}^m \vdash t : n}{\Gamma \vdash \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t : n}$$

$$\frac{\Gamma, \vec{x}^m \vdash t : n}{\Gamma \vdash \lambda \vec{x}.t : m \Rightarrow n} \qquad \frac{\Gamma \vdash s : m \Rightarrow n \quad \Gamma \vdash t : m}{\Gamma \vdash st : n}$$

$$\frac{\Gamma, \vec{x}_i^{m_i} \vdash s_i : n_i\ (1 \le i \le r) \quad \Gamma \vdash t : m}{\Gamma \vdash \mathsf{K}((\vec{x}_1)s_1, \ldots, (\vec{x}_r)s_r; t) : n} \qquad \frac{\Gamma, x^p, y^q, \Gamma' \vdash t : m}{\Gamma, y^q, x^p, \Gamma' \vdash t : m}$$

where, in the derivation of $\mathsf{K}(\ldots)$, K has arity rule $((m_1, n_1), \ldots, (m_r, n_r)) \to (m, n)$. □

Definition B.3 The *equality judgement* $\Gamma \vdash s = t : n$, where $\Gamma \vdash s : n$ and $\Gamma \vdash t : n$, is defined as a congruence relation on well-typed terms of the same arity under the same context generated from the following axioms.

$$\begin{array}{lcl}
\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ \vec{x} & = & t \\
\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ (\mathsf{letrec}\ \vec{y}\ \mathsf{be}\ s\ \mathsf{in}\ t)\ \mathsf{in}\ u & = & \mathsf{letrec}\ \vec{x}, \vec{y}\ \mathsf{be}\ t \otimes s\ \mathsf{in}\ u \\
\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ \mathsf{letrec}\ \vec{y}\ \mathsf{be}\ t\ \mathsf{in}\ u & = & \mathsf{letrec}\ \vec{x}, \vec{y}\ \mathsf{be}\ s \otimes t\ \mathsf{in}\ u \\
s \otimes (\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ u) & = & \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ s \otimes u \\
(\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t) \otimes u & = & \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t \otimes u \\
\mathsf{letrec}\ \vec{x}, \vec{y}, \vec{z}\ \mathsf{be}\ s_1 \otimes s_2 \otimes s_3\ \mathsf{in}\ t & = & \mathsf{letrec}\ \vec{y}, \vec{x}, \vec{z}\ \mathsf{be}\ s_2 \otimes s_1 \otimes s_3\ \mathsf{in}\ t \\
\mathsf{K}((\vec{x}_1)t_1, \ldots, (\vec{x}_r)t_r; \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t) & = & \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ \mathsf{K}((\vec{x}_1)t_1, \ldots, (\vec{x}_r)t_r; t) \\
 & & \\
(\lambda \vec{x}.s)t & = & \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t \\
\lambda \vec{x}.y\vec{x} & = & y \\
(\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ t)u & = & \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ s\ \mathsf{in}\ tu \\
s(\mathsf{letrec}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ u) & = & \mathsf{letrec}\ \vec{x}\ \mathsf{be}\ t\ \mathsf{in}\ su \\
 & & \\
\mathsf{letrec}\ x, \vec{y}\ \mathsf{be}\ v \otimes s\ \mathsf{in}\ t & = & \mathsf{letrec}\ x, \vec{y}\ \mathsf{be}\ v \otimes (s\{v/x\})\ \mathsf{in}\ t\{v/x\} \\
\mathsf{letrec}\ x\ \mathsf{be}\ v\ \mathsf{in}\ s & = & s \quad (x \notin fv(s) \cup fv(v))
\end{array}$$

□