The Safe Lambda Calculus

William Blum

Linacre College

Submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Oxford University Computing Laboratory

Michaelmas 2008

Abstract

We consider a syntactic restriction for higher-order grammars called *safety* that constrains occurrences of variables in the production rules according to their type-theoretic order. We transpose and generalize this restriction to the setting of the simply-typed lambda calculus, giving us what we call the *safe lambda calculus*. We study the expressivity of the calculus and show a result in the same vein as Schwichtenberg's 1976 characterization of the simply-typed lambda calculus: we show that the numeric functions representable in the safe lambda calculus are exactly the multivariate polynomials; thus conditional is not definable. We also give a characterization of representable word functions. We then study the complexity of deciding beta-eta equality of two safe simply-typed terms and show that this problem is PSPACE-hard. The safety restriction is then extended to other applied lambda calculi featuring recursion and references such as PCF and Idealized Algol (IA for short).

In order to study the game semantics of safe languages, we introduce a new concrete presentation of game semantics based on the theory of *traversals*: we show that the *revealed game denotation* of a term can be computed by traversing some souped-up version of the abstract syntax tree of the term using adequately defined traversal rules. This result was presented at the Galop workshop at ETAPS 2008. This allows us to give a game-semantic analysis of safety via syntactic reasoning: we show that safe lambda-terms are denoted by what we call *P-incrementally justified strategies*. This result was presented at TLCA 2007.

Next we study models of the safe lambda calculus and show that these are captured by *Incremental Closed Categories*. We build a categorical game model of the safe lambda calculus which gives rise to a fully abstract game model of safe IA. The model obtained for safe IA is effectively presentable: two terms are equivalent just if they have the same set of complete *O-incrementally justified* plays, where O-incremental justification is defined as the dual of P-incremental justification.

Finally in the last chapter we study safety from the point of view of algorithmic game semantics. We observe that up to the 3^{rd} order, the addition of unsafe context is conservative for observational equivalence (for both IA and safe IA). This implies that all the upper complexity bounds known for the lower-order fragments of IA also hold for the safe fragment; we show that it is also the case for the known lower-bounds. At order 4, observational equivalence was shown to be undecidable for IA. We conjecture that for the order-4 *safe* fragment of IA, the problem is reducible to the DPDA-equivalence problem (which is decidable).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The *safety condition* was introduced by Knapik, Niwiński and Urzyczyn at FoSSaCS 2002 [KNU02] in a seminal study of the algorithmics of infinite trees generated by higher-order grammars. The idea, however, goes back some twenty years to Damm [Dam82] who introduced an essentially equivalent[1] syntactic restriction (for generators of word languages) in the form of *derived types*. A higher-order grammar (that is assumed to be *homogeneously typed*) is said to be *safe* if it obeys certain syntactic conditions that constrain the occurrences of variables in the production (or rewrite) rules according to their type-theoretic order. Though the formal definition of safety is somewhat intricate, the condition itself is manifestly important. As we survey in the following, higher-order *safe* grammars capture fundamental structures in computation, offer clear algorithmic advantages, and lend themselves to a number of compelling characterizations:

- *Word languages.* Damm and Goerdt [DG86] have shown that the word languages generated by order-n *safe* grammars form an infinite hierarchy as n varies over the natural numbers. The hierarchy gives an attractive classification of the semi-decidable languages: levels 0, 1 and 2 of the hierarchy are respectively the regular, context-free, and indexed languages (in the sense of Aho [Aho68]), although little is known about higher orders.

  Remarkably, for generating word languages, order-n *safe* grammars are equivalent to order-n pushdown automata [DG86], which are in turn equivalent to order-n indexed grammars [Mas74, Mas76].

- *Trees.* Knapik et al. have shown that the Monadic Second Order (MSO) theories of trees generated by *safe* (deterministic) grammars of every finite order are decidable[2].

  They have also generalized the equi-expressivity result due to Damm and Goerdt [DG86] to an equivalence result with respect to generating trees: A ranked tree is generated by an order-n *safe* grammar if and only if it is generated by an order-n pushdown automaton.

- *Graphs.* Caucal [Cau02] has shown that the MSO theories of graphs generated[3] by *safe* grammars of every finite order are decidable. In a recent paper [HMOS08], however, Hague et al. have shown that the MSO theories of graphs generated by order-n *unsafe* grammars are undecidable, but deciding their modal mu-calculus theories is n-EXPTIME complete.

[1] See de Miranda's thesis [dM06] for a proof.

[2] It has recently been shown [Ong06a] that trees generated by *unsafe* deterministic grammars (of every finite order) also have decidable MSO theories. More precisely, the MSO theory of trees generated by order-n recursion schemes is n-EXPTIME complete.

[3] These are precisely the configuration graphs of higher-order pushdown systems.

1.2 Overview

The aim of this thesis is to understand the safety condition in the setting of the typed lambda calculus. Our first task is to transpose it to the lambda calculus and pin it down as an appropriate sub-system of the simply-typed theory. A first version of the *safe lambda calculus* has appeared in an unpublished technical report [AdMO04]. Here we propose a more general and cleaner version where terms are no longer required to be homogeneously typed. The formation rules of the calculus are designed to maintain a simple invariant: Variables that occur free in a safe lambda-term have orders no smaller than that of the term itself. We can now explain the sense in which the safe lambda calculus is safe by establishing its salient property: No variable capture can ever occur when substituting a safe term into another. In other words, in the safe lambda calculus, it is *safe* to use capture-*permitting* substitution when performing β-reduction.

There is no need for new names when computing β-reductions of safe lambda-terms, because one can safely "reuse" variable names in the input term. Safe lambda calculus is thus cheaper to compute in this naïve sense. Intuitively one would expect the safety constraint to lower the expressivity of the simply-typed lambda calculus. Our next contribution is to give a precise measure of the "expressivity deficit" of the safe lambda calculus. An old result of Schwichtenberg [Sch76] says that the numeric functions representable in the simply-typed lambda calculus are exactly the multivariate polynomials *extended with the conditional function*. In the same vein, we show that the numeric functions representable in the safe lambda calculus are exactly the multivariate polynomials.

> **Theorem 3.59** *The numeric functions (Church-)representable in the safe lambda calculus are exactly the multivariate polynomials.*

We further obtain a similar characterization concerning representable word functions.

> **Theorem 3.65** *The word functions definable in the safe lambda calculus are given by the minimal set containing (a) concatenation, (b) substitution, (c) the projections, (d) the constant functions; and closed by composition.*

In order to get a better understanding of our calculus, it is interesting to recast common problems studied in the literature on the simply-typed lambda calculus in the setting of the safe lambda calculus. We show for instance that the type-checking and typability problems remain decidable. We also consider the type-inhabitation problem: "Is there a term inhabiting a given type?". This problem is already relatively complex in the simply-typed lambda calculus—Statman showed that it is PSPACE-complete. Because of the somewhat intricate way in which safety constrains the occurrences of the variables, the inhabitation problem becomes even more complex in the safe lambda calculus. We do not know whether the problem is decidable.

Another famous result by Statman is that deciding beta-equality of two simply-typed terms is non-elementary. There are several proofs of this result in the literature. All of them proceed by reduction of a non-elementary problem—such as quantifier elimination in finite type theory—into the simply-typed lambda calculus. Interestingly, all these encodings make use of unsafe terms in some place. This suggests that such encoding is impossible in the safe lambda calculus and that the beta-equivalence problem may be simpler when restricted to safe terms. The author has not been able to establish an upper-bound on the complexity of this problem but a lower-bound is provided: the True Quantifier Boolean Formula (TQBF) problem (*i.e.*, deciding whether a quantified boolean formula is true) can be encoded in the safe lambda calculus. Since the latter problem is PSPACE-complete, this implies:

> **Theorem 3.55** *The beta-equivalence problem for safe lambda-terms is PSPACE-hard.*

A particularity of this encoding is that it relies on the entire type hierarchy and thus we only have PSPACE-hardness for the safe lambda calculus in its entirety. This contrasts with another result by Statman which says that there exists a finite set of types such that the beta-eta equivalence problem restricted to simply-typed terms of these types is PSPACE-hard.

Extensions

PCF is the simply-typed lambda calculus augmented with basic arithmetic operators, if-then-else branching and a family of recursion combinator Y_A of type $(A \to A) \to A$ for every type A. We define *safe* PCF to be the fragment of PCF obtained by constraining the application and abstraction rules in the same way as the safe lambda calculus. This language inherits the good properties of the safe lambda calculus: No variable capture occurs when performing substitution and safety is preserved by the reduction rules of the small-step semantics of PCF. Similarly, we define safe IA as safe PCF augmented with the imperative features of Idealized Algol (IA for short) [Rey81]. A version of the no variable capture lemma also holds in safe IA.

A concrete game semantics

Game semantics has emerged as a powerful paradigm for the study of higher-order functional programming languages in general, and in particular for the mother of all functional languages: the lambda calculus. The game approach was for instance the first to give rise to a fully abstract model of PCF [AMJ94, HO00].

In the traditional presentation of game semantics, attention is taken to abstract away entirely the syntax of the language from the definition of the semantics. This syntax-independent aspect of game models constitutes their salient feature. A substantial part of the thesis is devoted to giving a presentation of game semantics that is more concrete than the traditional one in the sense that the semantic denotation of a term carries some information about its syntax. This presentation is based on ideas recently introduced by Ong [Ong06a]: A term is canonically represented by a certain abstract syntax tree of its η-long normal form referred as the *computation tree*. A computation is then described by a justified sequence of nodes of the computation tree respecting some formation rules and called a *traversal*. Essentially, traversals allow us to model β-reductions without altering the structure of the computation tree via substitution. A notable property is that *P-views* (in the game-semantic sense) of traversals correspond to paths in the computation tree. We show that traversals are just representations of the *revealed game semantic* denotation (*i.e.*, the set of uncoverings of plays of the game-semantic denotation with respect to the syntax of the eta-long normal form). The standard game denotation can then be recovered by extracting the *cores* of the traversals, an operation that eliminates nodes that are "internal" to the computation—the counterpart of the hiding operation of game semantics. This leads to an isomorphism between the standard strategy denotation of a term and the set of traversal cores of its computation tree:

> **Theorem 4.96** (The Correspondence Theorem) *The set of traversals of the computation tree of a simply-typed term-in-context $\Gamma \vdash M : T$ is isomorphic to its revealed denotation $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$; the set of traversal cores is isomorphic to the standard game denotation $[\![\Gamma \vdash M : T]\!]$.*

We then extend our presentation of game semantics to PCF and Idealized Algol (PCF extended with block-allocated variables). We extend the notion of computation tree to recursively defined terms as follows: The computation tree of a PCF term is defined as the least upper-bound of the chain of computation trees of its *syntactic approximants* [AM98b]. Think of it as the tree obtained by expanding Y combinators *ad infinitum*. For instance the computation tree of $\mathsf{Y}(\lambda f x. f x)$ is given by the abstract syntax tree of the η-long normal form of the infinite lambda-term $(\lambda f x. f x)((\lambda f x. f x)((\lambda f x. f x)(\ldots$ It is possible to define traversal rules modeling the arithmetic constants of PCF so that a version of the Correspondence Theorem holds for PCF.

The extension to IA is complicated by the presence of the base type `var` used for reference variables. Indeed, the game denotation of `var` has infinitely many initial moves, therefore there is a mismatch between the tree representation of a term of type `var` and the arena underlying the game induced by the type `var`. It is possible, however, to adapt the game-semantic correspondence to IA by generalizing the notion of computation tree to computation *hypertrees* (*i.e.*, trees in which sibling nodes can be grouped together into a single *hypernode*).

On a more applied side, I have implemented a tool to illustrate the theory of traversals and its correspondence with game semantics [Blu08].

Game semantics of safety

A question inevitably arising is: Does the safety constraint noticeably impact on the game denotation of a term? Answering this question can help us gain a better understanding of the fundamental nature of the safety restriction. Using the correspondence between the game semantics of a lambda-term M and the set of traversals over its computation tree, we give a game-semantic characterization of safety. We show that the safety syntactic restriction is semantically captured by the *P-incrementally justified strategies*:

> **Theorem 5.19** *Let* $\vdash_{\mathsf{st}} M : A$ *be a closed simply-typed term. Then*
>
> $$M \textit{ has a safe } \beta\textit{-normal form} \iff [\![\vdash M : A]\!] \textit{ is P-incrementally justified strategies.}$$

In a *P-incrementally justified strategy*, pointers emanating from the P-moves of a play are uniquely reconstructible from the underlying sequence of moves and the pointers associated to the O-moves therein. More precisely, a strategy is *P-incrementally justified* just if each P-question in a play points to the last pending O-question of greater order in the P-view at that point. Thus up to order 3, pointers are superfluous in the game semantics of safe lambda-terms; from order 4 onwards, they are only necessary for O-questions.

A model of safe lambda calculi

Our last contribution is to establish a game model of the safe lambda calculus. A fundamental result in theoretical computer science is the connection between Cartesian Closed Categories (CCC) and models of typed lambda calculi: it was observed by Lambek [Lam86] that any extensional model of the simply-typed lambda calculus is a CCC, and conversely, any typed lambda calculus induces a CCC.

A similar categorical connection can be made for models of the safe lambda calculus. The categorical counterparts of safe lambda calculi are the *Incremental Closed Categories* (ICC). These categories are subcategories of CCC in which *currying* is restrained. By showing that P-incrementally justified strategies compose, we can construct an ICC of games with morphisms given by P-incrementally justified strategies. This gives rise to a categorical game model of the safe lambda calculus:

> **Proposition 6.50** *There is a Incremental Closed Category with games as objects and (closed) P-incrementally justified strategies as morphisms that soundly models the safe lambda calculus.*

Full abstraction

A common concept in game semantics is that the pure functional core of a programming language can be modeled by strategies satisfying the properties of *visibility*, *innocence* and *well-bracketing*. Adding features to the language corresponds to relaxing one of these properties in the game model. For instance adding imperative features breaks innocence, adding non-local control breaks well-bracketing and adding general references breaks visibility. Furthermore in each of these cases, the game model gives rise to a fully abstract model of the considered language. For instance the well-bracketed and visible strategies give rise to a fully abstract game model of the language Idealized Algol (IA). These results are summarized in Table 1.1.

Conversely, restricting the language corresponds to imposing more constraints on the strategy. The strategy counterpart of the safety restriction is P-incremental justification (P-i.j. for short). In this thesis we show that this restriction gives rise to a fully-abstract model for the safe fragment of PCF and IA:

Theorem 6.79 and 6.85 (Full abstraction) *Two safe (PCF or IA) terms are observationally equivalent* with respect to safe contexts *if and only if their denotations are equivalent with respect to the intrinsic preorder of the ICC game model.*

Language	Strategy constraints	Reference
Safe IA	D + V + B + P-i.j.	Theorem 6.85
Safe PCF	D + V + B + I + P-i.j.	Theorem 6.79
PCF	D + V + B + I	[HO00]
IA (PCF + local state)	D + V + B	[AM97, AM99]
PCF + non local control (call/cc)	D + V + I	[Lai97]
Core ML + general references	D + B	[AHM98]

where 'D', 'V', 'B', 'I' stand for 'deterministic', 'visible', 'well-bracketed' and 'innocent' respectively.

Table 1.1: Summary of Full Abstraction results.

Algorithmic game semantics

The game-semantic approach has become a very successful paradigm after the resolution of the long-standing full abstraction problem of PCF. For instance, researchers have been able to use game semantics to derive decision procedures for the observational equivalence problem (Given two terms, can they be used interchangeably?)—a research activity known as *Algorithmic game semantics*. A major breakthrough was the Characterization Theorem [AM97]: observational equivalence of two Idealized Algol terms is characterized by equality of the set of complete plays of their denotation. (Consequently, the game model of Idealized Algol is effectively presentable—a property that is not enjoyed by any model of PCF [Loa01].) This result paved the way to interesting characterizations of the game denotation of lower-order IA terms. Ghica and McCusker observed [GM00] that pointers are unnecessary for representing plays in the game semantics of the second-order finitary fragment of Idealized Algol (IA_2 for short). Consequently observational equivalence for this fragment can be reduced to the problem of equivalence of regular expressions. Similar characterizations were later obtain for other finitary fragments. For instance at order 3, although pointers are necessary, deciding observational equivalence of IA_3 is EXPTIME-complete [Ong04, MW05]. These results are all based on the same observation: At lower orders, the justification pointers present in the game denotation are either not required (*e.g.*, at order 2) or can be encoded succinctly (*e.g.*, at order 3). The possibility of representing plays *without some or all of their pointers* under the safety assumption strongly suggests that similar result can be obtained for the safe fragment of IA.

Our last contribution consists in studying the safety from the point of view of algorithmic game semantics. We introduce a new notion of observational equivalence for IA: A *safe context* is a safe IA term-in-context with a hole (a distinguished variable occurring exactly once in the term); two terms are considered equivalent if no safe context can distinguish them. We show that up to order 3 this notion of observational equivalence coincides with the usual one. A basic result in algorithmic game semantics is the Characterization Theorem: Observational equivalence of two IA terms is characterized by the equality of their set of complete plays. We show a version of this theorem for our notion of observational equivalence:

> **Theorem 6.87** (Characterization Theorem) *Two terms are observationally equivalent with respect to* safe contexts *if and only if they have the same set of P-incremental justified complete plays.*

Finally, based on these results, we show that all the known results [GM00, Ong02, MW05, MOW05, Mur03, Mur05a] about the complexity of observational equivalence up to order 3 are also valid for our new notion of observational equivalence:

> **Theorem** *(Sec. 6.6) The observational equivalence problem (with respect to safe contexts) for the safe finitary fragment of*
>
> *(a) order-*2 *IA + iteration is in PSPACE;*
>
> *(b) order-*2 *IA + order-*1 *recursion is undecidable;*
>
> *(c) order-*3 *+ iteration is EXPTIME-complete;*
>
> *(d) order-*3 *+ ground type recursion is reducible to the equivalence problem for deterministic pushdown automata (DPDA), and is thus decidable.*

This suggests that the restriction imposed on contexts kicks in at order-4. Murawksi has shown that the problem for (not necessarily safe) terms is undecidable at order-4 [Mur03, Mur05a]. His proof can be reused to show that the observational equivalence problem for safe order-4 terms and unrestricted (*i.e.*, not necessarily safe) contexts remains undecidable. We further make the following conjecture:

> **Conjecture 6.93** *The observational equivalence problem for* safe terms *with respect to* safe contexts *reduces to the DPDA-equivalence problem and is thus decidable.*

1.3 Organization of the thesis

The next chapter lays down the background for the rest of the thesis. It introduces briefly the simply-typed lambda calculus and two of its extensions that will be studied throughout the thesis, namely PCF and Idealized Algol. It then presents *higher-order grammars*, the original setting in which the safety restriction firstly appeared, and presents the safety restriction with some related results. Finally, the last section is devoted to the presentation of the basics and main results of game semantics. It also fixes notations that will be used in other chapters.

Chapter 3 introduces the definition of the *safe lambda calculus*. It establishes basic properties of the calculus and gives an account of its expressivity and complexity. The chapter concludes with a generalization of the safety restriction to other applied lambda calculi such as PCF and Idealized Algol.

Chapter 4 takes a detour from the safety restriction. It presents and extends the theory of traversals originally introduced by Ong [Ong06a]. It defines the notions of *computation tree* of a simply-typed term and *traversals* over this tree. Ultimately we prove the *Correspondence Theorem*, an important result that establishes a correspondence between traversals of the computation tree and the game-semantic denotation of a term.

In **Chapter 5**, we use this correspondence theorem to give an account of the game semantics of safety using a very simple syntactic argument.

Chapter 6 is concerned with categorical models for the safe lambda calculus, safe PCF and safe Idealized Algol. We derive a game model that is fully abstract for safe PCF and safe IA and we consider applications to algorithmic game semantics.

1.4 Authorship

Chapter 3, 4 and 5 are based on joint work with my supervisor. A summary of the work of Chapter 3 has appeared as an extended abstract in TLCA [BO07], a journal version is in press. A paper based on Chapter 4 will be submitted elsewhere. Chapter 6 is solely my work.

Chapter 2

Background

This chapter introduces in three sections the basic concepts that will be used throughout the thesis. The first section presents the lambda calculus; the second gives a brief introduction to higher-order grammars and presents the original definition of the safety restriction; the last section is a condensed account of game semantics.

2.1 Lambda Calculus

We assume that the reader is familiar with the simply-typed lambda calculus, but for precision and to fix notations we give here a brief overview of the basic definitions. For a detailed account the reader is referred to the standard textbooks on the subject [Hin97, HS86, Bar92].

2.1.1 Terms

We fix a countable set of variables $\mathcal{V}$.

Definition 2.1. The set Λ of ***terms*** of the *untyped lambda calculus* is given by the set of derivations of the following grammar:

$$\Lambda = \mathcal{V} \mid \Lambda\,\Lambda \mid \lambda\mathcal{V}.\Lambda \ .$$

These three basic formation rules are used to construct terms that are respectively *variables*, *applications* and *lambda-abstractions*.

A term is represented by an expression representing its derivation tree. It is computed as follows: The leaves of the derivation tree are concatenated from left to right and additional parentheses are added to indicate the order of the derivation. Parentheses ensure that the representation is unique. For instance they allow us to distinguish the five different derivations whose underlying concatenation of leaves is given by "$\lambda x.MNQ$"; these derivations are $\lambda x.((MN)Q)$, $\lambda x.(M(NQ))$, $(\lambda x.M)(NQ)$, $(\lambda x.(MN))Q$, and $((\lambda x.M)N)Q$. We further use the following conventions:

(i) We use symbols $x, y, \ldots$ to denote variables in $\mathcal{V}$ and $M, N, \ldots$ to denote other terms;

(ii) Application associates to the left: MNQ stands for the term $((MN)Q)$;

(iii) Nested lambda abstractions are combined into a single one: $\lambda xyz.x$ stands for $\lambda x.\lambda y.\lambda z.x$. Also if $\overline{x}$ denotes a sequence of variables $x_1 \ldots x_n$ then we write $\lambda\overline{x}.M$ as a short-hand for $\lambda x_1 \ldots x_n.M$.

Example 2.2. $\lambda x.x$, $\lambda x.xy$, $(\lambda x.xx)(\lambda x.xx)$ are all valid terms.

Definition 2.3. The set of ***free variables*** $FV(M)$ of a term M is given inductively by:

$$
\begin{aligned}
FV(x) &= \{x\} \\
FV(MN) &= FV(M) \cup FV(N) \\
FV(\lambda x.M) &= FV(M) \setminus \{x\} \ .
\end{aligned}
$$

An *occurrence* of a variable x in M is said to be ***free*** if it belongs to $FV(M)$. Otherwise it is said to be ***bound***. A term M is ***closed*** if it has no free variable (*i.e.*, $FV(M) = \emptyset$).

We write $\mathsf{closure}(M)$ to denote the closed term obtained from M by abstracting all its free variables by order of appearance in the term.

A variable is ***fresh*** if it does not occur anywhere in the terms that we are considering. Two terms M and N are ***α-convertible*** if one can be obtained from the other by renaming bound variables to fresh names. We consider syntactic equality of terms modulo α-conversion and we write $M \equiv N$ to denote this equality.

The set $\mathrm{sub}(M)$ of ***sub-terms*** of M is given by induction as:

$$
\begin{aligned}
\mathrm{sub}(x) &= \{x\} \\
\mathrm{sub}(MN) &= \{MN\} \cup \mathrm{sub}(M) \cup \mathrm{sub}(N) \\
\mathrm{sub}(\lambda x.M) &= \{\lambda x.M\} \cup \mathrm{sub}(M) \ .
\end{aligned}
$$

2.1.2 Substitution

Substitution refers to the transformation that replaces a free variable in a term by another term. The naive way to implement substitution consists in textually replacing all free occurrences of x in M by N. This is called *capture-permitting substitution*:

Definition 2.4. The ***capture-permitting substitution*** of N for x in M, written $M\{N/x\}$, is defined by induction as follows:

$$
\begin{aligned}
x\{N/x\} &\equiv N \\
y\{N/x\} &\equiv y \quad \text{if } x \not\equiv y, \\
(M_1 M_2)\{N/x\} &\equiv (M_1\{N/x\})(M_2\{N/x\}) \\
(\lambda x.M)\{N/x\} &\equiv \lambda x.M \\
(\lambda y.M)\{N/x\} &\equiv \lambda y.M\{N/x\} \text{ if } y \not\equiv x \quad .
\end{aligned}
$$

Although this definition is valid, it is not adequate in the sense that is not sound with respect to syntactical equality: take the terms $M_1 \equiv \lambda y.x$, $M_2 \equiv \lambda z.x$ and $N \equiv y$. We have $M_1\{N/x\} \equiv \lambda y.y$ and $M_2\{N/x\} \equiv \lambda z.N$. Thus although M_1 and M_2 are syntactically equivalent, performing the same substitution on both terms gives terms that are not syntactically equivalent.

The source of the problem lies the last equation: in the abstraction case, when pushing the substitution under the lambda, some care needs to be taken so that the free-variables in M do not get "captured" by the abstraction. This is traditionally achieved by renaming all the free variables in M afresh before continuing with the substitution:

Definition 2.5. The ***substitution*** of N for x in M written $M[N/x]$ is defined by induction as follows:

$$
\begin{aligned}
x[t/x] &\equiv t \\
y[t/x] &\equiv y \quad \text{if } x \not\equiv y, \\
(M_1 M_2)[t/x] &\equiv (M_1[t/x])(M_2[t/x]) \\
(\lambda x.M)[t/x] &\equiv \lambda x.M \\
(\lambda y.M)[t/x] &\equiv \lambda z.M[z/y][t/x] \text{ if } x \not\equiv y \text{ and where } z \text{ is a fresh variable.}
\end{aligned}
$$

Observe that only the last equation differs from the previous definition.

The obvious generalization to multiple variables is called *simultaneous substitution*:

Definition 2.6. The ***simultaneous capture-permitting substitution*** of $N_1, \ldots, N_n$ for the (distinct) variables $x_1, \ldots x_n$ in M, written $M\{N_1/x_1, \ldots, N_n/x_n\}$ and abbreviated here as $M\left\{\overline{N}/\overline{x}\right\}$ is defined by induction as follows:

$$\begin{array}{rcl}
x_i\left\{\overline{N}/\overline{x}\right\} & \equiv & N_i \\
y\left\{\overline{N}/\overline{x}\right\} & \equiv & y \quad \text{if } x \neq y_i \text{ for all } i, \\
(M_1M_2)\left\{\overline{N}/\overline{x}\right\} & \equiv & (M_1\left\{\overline{N}/\overline{x}\right\})(M_2\left\{\overline{N}/\overline{x}\right\}) \\
(\lambda x_i.M)\left\{\overline{N}/\overline{x}\right\} & \equiv & \lambda x_i.M\left\{N_1 \ldots N_{i-1}N_{i+1} \ldots N_n/x_1 \ldots x_{i-1}x_{i+1} \ldots x_n\right\} \\
(\lambda y.M)\left\{\overline{N}/\overline{x}\right\} & \equiv & \lambda y.M\left\{\overline{N}/\overline{x}\right\} \text{if } y \not\equiv x_i \text{ for all } i.
\end{array}$$

Definition 2.7. The ***simultaneous substitution*** of $N_1, \ldots, N_n$ for the (distinct) variables $x_1, \ldots x_n$ in M, written $M[N_1/x_1, \ldots, N_n/x_n]$ and abbreviated here as $M\left[\overline{N}/\overline{x}\right]$ is defined by induction as follows:

$$\begin{array}{rcl}
x_i\left[\overline{N}/\overline{x}\right] & \equiv & N_i \\
y\left[\overline{N}/\overline{x}\right] & \equiv & y \quad \text{if } y \not\equiv x_i \text{ for all } i, \\
(MN)\left[\overline{N}/\overline{x}\right] & \equiv & (M\left[\overline{N}/\overline{x}\right])(N\left[\overline{N}/\overline{x}\right]) \\
(\lambda x_i.M)\left[\overline{N}/\overline{x}\right] & \equiv & \lambda x_i.M\left[N_1 \ldots N_{i-1}N_{i+1} \ldots N_n/x_1 \ldots x_{i-1}x_{i+1} \ldots x_n\right] \\
(\lambda y.M)\left[\overline{N}/\overline{x}\right] & \equiv & \lambda z.M[z/y]\left[\overline{N}/\overline{x}\right] \\
& & \text{if } y \not\equiv x_i \text{ for all } i \text{ and where } z \text{ is a fresh variable.}
\end{array}$$

2.1.3 Conversion

A binary relation R over Λ is ***compatible*** if $M\,R\,M'$ implies $MN\,R\,M'N$, $NM\,R\,NM'$ and $\lambda x.M\,R\,\lambda x.M'$ for all $M, M', N \in \Lambda$. It is ***transitive*** if $M\,R\,N$ and $N\,R\,Q$ implies $M\,R\,Q$; ***reflexive*** if $M\,R\,M$; and ***symmetric*** if $M\,R\,N$ implies $N\,R\,M$, for all $M, N, Q \in \Lambda$.

The concept of computation in the lambda calculus is incarnated by a term-rewriting rule called ***β-reduction***:

Definition 2.8. We call ***β-redex*** any term of the form $(\lambda x.M)N$. Its *contraction* is defined as $M[N/x]$. We define β as the relation mapping a redex to its contraction:

$$\beta = \{((\lambda x.M)N,\ M[N/x])\ |M, N \in \Delta, x \in \mathcal{V}\}\ .$$

The ***one-step β-reduction*** relation $\rightarrow_\beta$ is defined as the compatible closure of the relation β. The relation $\twoheadrightarrow_\beta$ denotes the reflexive transitive closure of $\rightarrow_\beta$, and the relation $=_\beta$, called ***β-equality*** or also β-conversion, denotes the reflexive symmetric transitive closure of $\rightarrow_\beta$.

In addition to the β-reduction rule the ***η-reduction*** $\rightarrow_\eta$ is defined as the smallest compatible relation satisfying:

$$\lambda z.Mz \rightarrow_\eta M \quad \text{if } z \notin FV(M)\ .$$

We define *η-conversion* $=_\eta$ as the reflexive symmetric transitive closure of $\rightarrow_\eta$.

Definition 2.9 (Normal form)**.** A term

(i) is a ***β-normal form***, β-nf for short, if it does not contain any β-redex;

(ii) *has* a β-normal form, or is ***normalizable***, if it is β-equal to a β-normal form;

(iii) is ***strongly normalizable*** if every sequence of reduction starting from it is finite (and therefore ends with a normal form).

The notions of η and $\beta\eta$-normal form are defined similarly.

2.1.4 Properties

A reduction is ***weakly normalizing*** if every term is normalizable and ***strongly normalizing*** if every term is strongly normalizable. The (untyped) lambda calculus is not even weakly normalizing with respect to β-reduction since for instance the term $\Omega \equiv (\lambda x.x\,x)(\lambda x.x\,x)$ β-reduces to itself.

The lambda calculus satisfies the so-called *Church-Rosser* theorem:

Theorem 2.10 (Church-Rosser Theorem)**.** *If $M \twoheadrightarrow_\beta N_1$ and $M \twoheadrightarrow_\beta N_2$ then for some N we have $N_1 \twoheadrightarrow_\beta N$ and $N_2 \twoheadrightarrow_\beta N$.*

This is sometimes summarized as "$\twoheadrightarrow_\beta$ satisfies the diamond property". A consequence of this theorem is that a term has at most one β-normal form. Furthermore:

Theorem 2.11 (Normalization Theorem [Bar84])**.** *The leftmost reduction strategy is normalizing (i.e., if M has a normal form then the reduction strategy consisting in contracting the leftmost redex leads to that normal form).*

2.1.5 Simple types

Simple types are objects that are constructed from atomic types using the function space arrow operator $\rightarrow$. Formally, we fix a set $\mathbb{A}$ of ***atomic types*** and we define the set $\mathbb{T}_\mathbb{A}$ of ***simple types*** over $\mathbb{A}$ as the set generated from the following grammar:

$$\mathbb{T}_\mathbb{A} ::= \mathbb{A} \mid \mathbb{T}_\mathbb{A} \rightarrow \mathbb{T}_\mathbb{A} \ .$$

We will use the Greek letter symbols α, β, ... to refer to atomic types and capital letters A, B, ... to refer to other types. We further assume that $\mathbb{A}$ has a distinguished atomic type denoted by the symbol o.

By convention, $\rightarrow$ associates to the right. Thus every type can be written as $A_1 \rightarrow \cdots \rightarrow A_n \rightarrow \alpha$ for some atomic type α, which we shall abbreviate to $(A_1, \cdots, A_n, \alpha)$ (in case $n = 0$, we identify (α) with α). The number n is called the ***arity*** of the type, it is written $arity(T)$ for every type T.

Convention 2.12 We use the following abbreviations for types:

(i) For every atom a and natural number $n \in \mathbb{N}$, we define the types n_a as follows: $0_a = a$ and $(n+1)_a = n_a \rightarrow a$;

(ii) For every types A, B and positive natural number $n > 0$, the type $A^n \rightarrow B$ is defined by induction as: $A^1 \rightarrow B = A \rightarrow B$ and $A^{n+1} \rightarrow B = A \rightarrow (A^n \rightarrow B)$. In other words: $A^n \rightarrow B = \overbrace{A \rightarrow \ldots \rightarrow A}^{n \text{ times}} \rightarrow B$;

The ***order*** of a type is given by $\operatorname{ord} \alpha = 0$ for every atomic type α and $\operatorname{ord}(A \rightarrow B) = \max(1 + \operatorname{ord} A, \operatorname{ord} B)$. We assume an infinite set of typed variables. The order of a typed term or symbol is defined to be the order of its type.

Definition 2.13 (Type substitution)**.** A ***type substitution*** is an expression $[T_1/a_1, \ldots, T_n/a_n]$ where $a_1, \ldots, a_n$ are distinct atomic types in $\mathbb{A}$ and $T_1, \ldots, T_n \in \mathbb{T}$.

For every type $T \in \mathbb{T}$ and type substitution $[T_1/a_1, \ldots, T_n/a_n]$ we define $T[T_1/a_1, \ldots, T_n/a_n]$ to be the type obtained from T by substituting T_1 for a_1, ..., T_n for a_n. The resulting type is called an ***instance of*** the type T.

2.1.6 Simply-typed lambda calculus *à la* Curry

There exist two styles of presentation of the simply-typed lambda calculus. In the Curry style, typing is implicit. This means that each untyped term is assigned either no type or infinitely many types. The other presentation, called Church style, makes the typing information explicit in the structure of the term by introducing type annotations in it. Thus terms of this system have a unique type. We present here the Curry version of the simply-typed lambda calculus.

We write $M : A$ to denote that the term M can be assigned the type $A \in \mathbb{T}$ in the typing-system. A set Γ of *typing assumptions* is a set of typing-assignments of the form $x : T$ where x is a variable in $\mathcal{V}$ and $T \in \mathbb{T}$. It is *consistent* if all the variables names are distinct (*i.e.*, each variable name is assigned a unique type). The underlying set of variable names is called the domain Γ and is written $dom(\Gamma)$. We will write $\Gamma, x : A$ to denote the set of typing assumptions $\Gamma \cup \{x : A\}$. We consider judgements of the form $\Gamma \vdash_{\mathrm{Cu}} M : A$ called ***terms-in-context*** where Γ is a *consistent* set of typing assumptions called the ***typing context***, A is a simple type and M is a term.

Definition 2.14. The ***simply-typed lambda calculus*** *à la* Curry, denoted by $\Lambda_{\rightarrow}^{\mathrm{Cu}}$, is defined as the set of terms-in-context of the form $\Gamma \vdash_{\mathrm{Cu}} M : A$ that are derivable from the variable, application and abstraction rules defined as follows:

$$\frac{}{\Gamma \vdash_{\mathrm{Cu}} x : A}\ x : A \in \Gamma \qquad \frac{\Gamma \vdash_{\mathrm{Cu}} M : A \rightarrow B \quad \Gamma \vdash_{\mathrm{Cu}} N : A}{\Gamma \vdash_{\mathrm{Cu}} M\,N : B} \qquad \frac{\Gamma, x : A \vdash_{\mathrm{Cu}} M : B}{\Gamma \vdash_{\mathrm{Cu}} \lambda x.M : A \rightarrow B}$$

Whenever the context is empty we just write $\vdash_{\mathrm{Cu}} M : A$ instead of $\emptyset \vdash_{\mathrm{Cu}} M : A$.

In the literature, the second and third rules are sometimes called the $\rightarrow$-elimination and $\rightarrow$-introduction rules respectively.

The notion of "derivability" used in the above definition can be made more precise: A *typing derivation* or ***typing deduction*** Δ of $\Lambda_{\rightarrow}^{\mathrm{Cu}}$ is a tree labelled by terms-in-context of the form $\Gamma \vdash_{\mathrm{Cu}} M : A$ where the leaves are axioms and the internal nodes are deduced from their children nodes using the rules of $\Lambda_{\rightarrow}^{\mathrm{Cu}}$. Each edge of the tree is labeled by the rule used to make the deduction. The root of the tree is called the *conclusion* of the derivation. The derivation tree is usually represented with leaves at the top and root at the bottom [Hin97]. Terms-in-context of the simply-typed lambda calculus are then defined as the set of conclusions of derivations in $\Lambda_{\rightarrow}^{\mathrm{Cu}}$.

An ***inhabitant*** of a type $T \in \mathbb{T}$ is a term $M \in \Lambda$ such that for some typing-context Γ we have $\Gamma \vdash_{\mathrm{Ch}} M : T$.

The type substitution operation from Def. 2.13 naturally extends to finite sequences of types, contexts, terms-in-context and deductions. For instance for every context Γ, type B and atomic type α we write $\Gamma\,[B/\alpha]$ to denote the context obtained by performing the substitution $[B/\alpha]$ on each type occurring in Γ.

We now recall some standard results:

Proposition 2.15 (Weakening)**.** *Suppose* $\Gamma \vdash_{\mathrm{Cu}} M : A$ *and* Γ' *is a typing-context with* $\Gamma \subseteq \Gamma'$ *then* $\Gamma' \vdash_{\mathrm{Cu}} M : A$.

Proposition 2.16 (Typability of subterms)**.** *Let* M' *be a subterm of* M*. Then if* $\Gamma \vdash_{\mathrm{Cu}} M : A$ *then* $\Gamma' \vdash_{\mathrm{Cu}} M' : A'$ *for some context* Γ' *and type* A'.

Lemma 2.17 (Substitution Lemma)**.**

(i) If $\Gamma, x : A \vdash_{\mathrm{Cu}} M : B$ *and* $\Gamma \vdash_{\mathrm{Cu}} N : A$ *then* $\Gamma \vdash_{\mathrm{Cu}} M\,[N/x] : B$;

(ii) If $\Gamma \vdash_{\mathrm{Cu}} M : A$ *then* $\Gamma\,[B/\alpha] \vdash_{\mathrm{Cu}} N : A\,[B/\alpha]$.

Theorem 2.18 (Subject Reduction)**.** *Suppose that* $M \twoheadrightarrow_{\beta} N$*. Then*

$$\Gamma \vdash_{\mathrm{Cu}} M : A \implies \Gamma \vdash_{\mathrm{Cu}} M' : A\ .$$

2.1.6.1 Typing problems

The three following problems are often considered in type theory:

- TYPE CHECKING: Given a term M, context Γ and type A, do we have $\Gamma \vdash_{\mathrm{Cu}} M : A$?
- TYPABILITY: Given a term M and context Γ, is there a type A such that $\Gamma \vdash_{\mathrm{Cu}} M : A$?
- INHABITATION: Given a type A, is there a term M such that $\vdash_{\mathrm{Cu}} M : A$?

Definition 2.19 (Principality)**.** A term M has ***principal type*** A if $\Gamma \vdash_{\mathrm{Cu}} M : A$ for some context Γ, and for every possible derivation $\Gamma' \vdash_{\mathrm{Cu}} M : A'$, A' is an instance of A. A ***principal deduction*** for a term M is a deduction Δ of the term-in-context $\Gamma \vdash_{\mathrm{Cu}} M : T$ such that every other deduction with the same conclusion is an instance of Δ, so in particular T is a *principal type* of M.

The principal type is unique up to variable renaming [Hin97].

Theorem 2.20 (PT Theorem, Curry, Hindley, Milner)**.** *It is decidable whether a term is typable in $\Lambda_{\to}$. Moreover if M is typable then it has a* ***principal deduction*** *that is computable from M.*

This implies that both TYPE CHECKING and TYPABILITY are decidable.

Theorem 2.21 (Strong normalization, Tait [Tai67])**.** *Every term that is typable in $\Lambda_{\to}$ is strongly normalizable (i.e., every reduction sequence leads to its (unique) normal form).*

Theorem 2.22 (Statman [Sta79a])**.** *The problem* INHABITATION *for types defined over an infinite number of atoms is PSPACE-complete (and thus decidable).*

2.1.7 Simply-typed lambda calculus *à la* Church

The simply-typed lambda calculus that we have introduced corresponds to the *Curry-style* version. There is another approach called the *Church-style* presentation in which variable binders are annotated with types[1]. The set of annotated-types $\Lambda_{\mathbb{T}}$ is formally given by the following grammar:

$$\Lambda_{\mathbb{T}} = \mathcal{V} \mid \Lambda_{\mathbb{T}}\,\Lambda_{\mathbb{T}} \mid \lambda_{\mathbb{T}}\mathcal{V} : \mathbb{T}.\Lambda_{\mathbb{T}} \ .$$

Observe that in the abstraction case, the binder is annotated with a type. This is the only difference with untyped terms from Λ. For every annotated term $M \in \Lambda_{\mathbb{T}}$, the *untyped term underlying* M, written $|M|$, is obtained by erasing all the type annotations from M.

We can now introduce new judgements of the form

$$\Gamma \vdash_{\mathrm{Ch}} M : A$$

where M ranges over *annotated terms* $\Lambda_{\mathbb{T}}$. The simply-typed lambda calculus *à la* Church, written $\Lambda_{\to}^{\mathrm{Ch}}$, is then given by the following typing system:

$$\frac{}{\Gamma \vdash_{\mathrm{Ch}} x : A}\ x : A \in \Gamma \qquad \frac{\Gamma \vdash_{\mathrm{Ch}} M : A \to B \quad \Gamma \vdash_{\mathrm{Ch}} N : A}{\Gamma \vdash_{\mathrm{Ch}} M\,N : B} \qquad \frac{\Gamma, x : A \vdash_{\mathrm{Ch}} M : B}{\Gamma \vdash_{\mathrm{Ch}} \lambda x^{A}.M : A \to B}$$

In contrast with the Curry version, terms of the Church typed lambda calculus have a unique type at most:

Proposition 2.23 (Uniqueness of types in $\Lambda_{\to}^{\mathrm{Ch}}$)**.** *If $\Gamma \vdash_{\mathrm{Ch}} M : T$ and $\Gamma \vdash_{\mathrm{Ch}} M : T'$ then $T = T'$. Further if $\Gamma \vdash_{\mathrm{Ch}} M : T$, $\Gamma \vdash_{\mathrm{Ch}} M' : T'$ and $M =_{\beta} M'$ then $T = T'$.*

[1]In fact in the original Church presentation, variable occurrences are also annotated. The version that we present here is sometimes called the Bruijn-style simply-typed lambda calculus. These two presentations are essentially equivalent.

The Curry-style and Church-style systems are related by the following result:

Proposition 2.24. *(i) Let $M \in \Lambda_{\mathbb{T}}$. Then $\Gamma \vdash_{\mathrm{Ch}} M : A \implies \Gamma \vdash_{\mathrm{Cu}} |M| : A$.*

(ii) Let $M \in \Lambda$. Then $\Gamma \vdash_{\mathrm{Cu}} M : A \implies \exists M' \in \Lambda_{\mathbb{T}}$ s.t. $\Gamma \vdash_{\mathrm{Ch}} M' : A \wedge |M'| = M$.

In particular this implies

Corollary 2.25. *Let $T \in \mathbb{T}$. Then T is inhabited in $\Lambda^{\mathrm{Ch}}_{\rightarrow}$ iff it is inhabited in $\Lambda^{\mathrm{Cu}}_{\rightarrow}$.*

Convention 2.26 In the rest of this thesis we will use judgements of the form $\Gamma \vdash_{\mathsf{st}} M : A$ to denote both *à la* Curry and *à la* Church terms-in-context: if M is an annotated term in $\Lambda_{\mathbb{T}}$ then the judgement stands for $\Gamma \vdash_{\mathrm{Ch}} M : A$ whereas if M is an untyped term in Λ then it stands for $\Gamma \vdash_{\mathrm{Cu}} M : A$.

2.1.8 Extensions

The simply-typed lambda calculus can be extended with a set of typed constants Ξ. To allow the use of constants, the syntax of Λ has an additional grammar rule: $\Lambda = \ldots \mid \Xi$. The typing system is also augmented with the rule

$$(\mathsf{const}) \ \frac{}{\vdash_{\mathrm{Cu}} f : A} \ f \in \Xi \ .$$

A new notion of reduction is defined to allow contraction of terms whose head occurrence is a Ξ-constant: Every constant c in Ξ comes with a rewriting function $f_c : \Lambda^k \rightarrow \Lambda$ for some $k \in \mathbb{N}$ determining the interpretation of the constant. We then extend the reduction relation of the lambda calculus with the rule $c\, M_1 \ldots M_k \rightarrow f_c(M_1, \ldots, M_k)$ for every closed normal forms $M_1, \ldots M_k$ and constant c.

2.1.9 PCF

The *Programming language for Computable Functions*, PCF for short, is a simple programming language based on the *Logic of Computable Functions* (LCF) devised by Dana Scott [Sco69]. It was introduced in a classical paper by Plotkin "LCF considered as a programming language" [Plo77]. PCF can be viewed as the Church-like simply-typed lambda calculus extended with arithmetic operators, conditional and recursion.

Syntax

The set of types is $\mathbb{T}_{\mathtt{exp}}$—the simple types generated from the atomic type `exp` of natural numbers. PCF terms are given by the grammar:

$$\begin{aligned} M ::= {} & x \mid \lambda x^A.M \mid MM \mid \\ & \mid n \mid \mathtt{succ}\ M \mid \mathtt{pred}\ M \\ & \mid \mathtt{cond}\ MMM \mid \mathsf{Y}_A\ M \end{aligned}$$

where x ranges over a set of countably many variables, n represents an integer constant ranging over the set of natural numbers, `succ` represents the successor function on integer, `pred` is the predecessor function, `cond` the conditional (*i.e.*, if-then-else branching) and $\mathsf{Y}_A : (A \rightarrow A) \rightarrow A$ for every type A is the recursion combinator.

The language is formally given by terms-in-context of the form $\Gamma \vdash M : A$ defined by induction over the rules of Table 2.1.

$$\text{(var)}\frac{}{x_1 : A_1, x_2 : A_2, \dots x_n : A_n \vdash x_i : A_i}\ i \in 1..n$$

$$\text{(app)}\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B} \qquad \text{(abs)}\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A.M : A \to B}$$

$$\text{(const)}\frac{}{\Gamma \vdash n : \mathtt{exp}} \qquad \text{(succ)}\frac{\Gamma \vdash M : \mathtt{exp}}{\Gamma \vdash \mathtt{succ}\ M : \mathtt{exp}} \qquad \text{(pred)}\frac{\Gamma \vdash M : \mathtt{exp}}{\Gamma \vdash \mathtt{pred}\ M : \mathtt{exp}}$$

$$\text{(cond)}\frac{\Gamma \vdash M : \mathtt{exp} \quad \Gamma \vdash N_1 : \mathtt{exp} \quad \Gamma \vdash N_2 : \mathtt{exp}}{\Gamma \vdash \mathtt{cond}\ M\ N_1\ N_2 : \mathtt{exp}} \qquad \text{(rec)}\frac{\Gamma \vdash M : A \to A}{\Gamma \vdash \mathsf{Y}_A M : A}$$

Table 2.1: Formation rules for PCF terms.

Example 2.27. The integer addition function is definable in PCF by:

$$\textsc{plus} \equiv \mathsf{Y}_{\mathtt{exp}\to\mathtt{exp}\to\mathtt{exp}}(\lambda p^{\mathtt{exp}\to\mathtt{exp}\to\mathtt{exp}} x^{\mathtt{exp}} y^{\mathtt{exp}}.\mathtt{cond}\ x\ y\ (p\ (\mathtt{pred}\ x)\ (\mathtt{succ}\ y)))$$

so that for terms M and N, if $M \Downarrow m$ and $N \Downarrow n$, $m, n \in \mathbb{N}$ then $\textsc{plus}\, M\ N \Downarrow m + n$.

Equality on integer is also definable by:

$$\begin{aligned}\textsc{Eq} = \mathsf{Y}(\lambda f^{\mathtt{exp}\to\mathtt{exp}\to\mathtt{exp}}\ x^{\mathtt{exp}}\ y^{\mathtt{exp}}.\ \mathtt{cond}\ a \\ (\mathtt{cond}\ b\ 1\ 0) \\ (\mathtt{cond}\ b\ 0\ (f\ (\mathtt{pred}\ a)\ (\mathtt{pred}\ b))))\end{aligned}$$

so that $\textsc{Eq}\, M\ N \Downarrow 1$ if M and N evaluate to the same value, and $\textsc{Eq}\, M\ N \Downarrow 0$ otherwise.

Operational semantics

The operational semantics of the language is given using a big-step style semantics. We call ***canonical form*** a term that is either a number or a function. Formally this is given by the grammar

$$V ::= n \mid \lambda x^A.M \quad .$$

The notation $M \Downarrow V$ means that the closed term M evaluates to the canonical form V. We write $M \Downarrow$ if the judgement $M \Downarrow V$ is valid for some canonical form V. The full operational semantics is given in Table 2.2. Since the evaluation rules are defined for closed terms only, the context Γ is omitted in the rules.

$$\frac{}{V \Downarrow V} \quad \text{provided that } V \text{ is in canonical form.}$$

$$\frac{M \Downarrow \lambda x^A.M' \quad M'[N/x] \Downarrow V}{MN \Downarrow V}$$

$$\frac{M \Downarrow n}{\mathtt{succ}\ M \Downarrow n+1} \qquad \frac{M \Downarrow n+1}{\mathtt{pred}\ M \Downarrow n} \qquad \frac{M \Downarrow 0}{\mathtt{pred}\ M \Downarrow 0}$$

$$\frac{M \Downarrow 0 \quad N_1 \Downarrow V}{\mathtt{cond}\ M N_1 N_2 \Downarrow V} \qquad \frac{M \Downarrow n+1 \quad N_2 \Downarrow V}{\mathtt{cond}\ M N_1 N_2 \Downarrow V}$$

$$\frac{M(\mathsf{Y}M) \Downarrow V}{\mathsf{Y}M \Downarrow V}$$

Table 2.2: Big-step operational semantics of PCF.

Case constructs

PCF is sometimes extended with a family of k-ary conditionals formed with the rule:

$$(\mathsf{case})\frac{\Gamma\vdash M:\mathtt{exp}\qquad \Gamma\vdash N_1:\mathtt{exp}\quad\ldots\quad\Gamma\vdash N_k:\mathtt{exp}}{\Gamma\vdash \mathtt{case}_k\, M\, N_1\, N_2\ldots N_k:\mathtt{exp}}$$

The resulting language is referred as PCF$_c$. Its operational semantics is given by that of PCF together with the rule:

$$\frac{M\Downarrow i\quad N_{i+1}\Downarrow V}{\mathtt{case}_k\ N\ N_1\ N_2\ \ldots\ N_k\ \Downarrow V}\ i\in\{0,\ldots,k-1\}.$$

Syntactic sugar

For every integer $k\in\mathbb{N}$ and term $M:\mathtt{exp}$ we write "$M+k$" as syntactic sugar for "PLUS $M\,k$". For every terms M, N_1 and N_2 of type $\mathtt{exp}$ we write "$N_1=N_2$" for "EQ N_1 N_2", "$N_1\neq N_2$" for "$\mathtt{cond}\,(\text{EQ}\ N_2\ N_2)\,1\,0$", and "$\mathtt{if}\,M\,\mathtt{then}\,N_1\,\mathtt{else}\,N_2$" for "$\mathtt{cond}\,M\,N_2\,N_1$". We will also use the construct

$$\begin{aligned}&\mathtt{match}\,M\,\mathtt{with}\\&\qquad a_1\to N_1\\&\qquad|\ \ldots\\&\qquad|\,a_q\to N_q\\&\qquad|\,\_\to R\end{aligned}$$

for distinct integers $a_1,\ldots a_q$, $q\geq 1$, as syntactic sugar for "$\mathtt{case}_m\ M\ N'_1\ldots N'_m$" where $m=1+\max_{1\leq i\leq q}a_i$ and for $1\leq j\leq m$, $N'_j\equiv N_i$ if $j=a_i$ for some $1\leq i\leq q$ and $N'_j\equiv R$ otherwise.

2.1.10 Idealized Algol

Idealized Algol, IA for short, is an extension of PCF introduced by J.C. Reynold [Rey81]. It adds imperative features such as local variables and sequential composition. Its types is given by the simple types over the basis $\{\mathtt{com},\mathtt{exp},\mathtt{var}\}$ where $\mathtt{com}$ denotes the type of commands and $\mathtt{var}$ the type of local variables.

The most basic command is given by the constant $\mathtt{skip}$ of type $\mathtt{com}$ which performs no computation. Commands can be composed using the sequential composition operator $\mathtt{seq}_A$ for every base type A. The sequential composition of two terms $N_0:\mathtt{com}$ and $N_1:A$ is given by the term $M\equiv\mathtt{seq}_A N_0 N_1:\mathtt{com}$ which is interpreted operationally as follows: N_0 is evaluated first and if it terminates then the term N_1 is evaluated. In the case where $A=\mathtt{exp}$, the result of the evaluation of N_1 is returned; otherwise $A=\mathtt{com}$ and the command N_1 is just evaluated after N_0 and the expression yields no result. Terms formed with the operator $\mathtt{seq}_{\mathtt{exp}}$ are called *active expressions*.

Local variables are declared using the $\mathtt{new}$ operator, their content is modified using $\mathtt{assign}$ and retrieved using $\mathtt{deref}$. Operationally, these variables behave like memory cells.

In addition to these local variables, IA features the so called "*bad variable construct*" $\mathtt{mkvar}$. This operator can be used to construct a special variable by specifying custom assignment and dereferencing functions. (This addition to the language may seem a little bit artificial but its presence has semantic importance.[2]) It takes two arguments: The first one, called the *acceptor*, is the function that is responsible of affecting a value to the variable. The second one is an expression that returns the value held by the variable. This mechanism is similar to the "set/get" object programming paradigm used by C++ programmers. An example of such

[2]McCusker showed that the standard game model of IA is only equationally fully abstract for the language without bad variables, whereas for full IA, it is also *inequationally* fully abstract [McC03].

variable is the term $\texttt{mkvar}\ (\lambda v.\texttt{skip})\ 0$. Variables created that way are called “bad variables” because they do not necessarily behave like a memory cell: reading the content of the variable does not necessarily gives you the last value that was written. For instance the variable defined above always yield 0 regardless of the value that was written to it previously.

The syntax

The typing system for IA is an extension of that of PCF. The additional rules are given in Table 2.3.

$$\frac{\Gamma \vdash M : \texttt{com} \quad \Gamma \vdash N : A}{\Gamma \vdash \texttt{seq}_A\, M\, N : A}\quad A \in \{\texttt{com}, \texttt{exp}\}$$

$$\frac{\Gamma \vdash M : \texttt{var} \quad \Gamma \vdash N : \texttt{exp}}{\Gamma \vdash \texttt{assign}\, M\, N\ : \texttt{com}} \qquad \frac{\Gamma \vdash M : \texttt{var}}{\Gamma \vdash \texttt{deref}\, M : \texttt{exp}}$$

$$\frac{\Gamma, x : \texttt{var} \vdash M : A}{\Gamma \vdash \texttt{new}\ x\ \texttt{in}\ M}\quad A \in \{\texttt{com}, \texttt{exp}\}$$

$$\frac{\Gamma \vdash M_1 : \texttt{exp} \to \texttt{com} \quad \Gamma \vdash M_2 : \texttt{exp}}{\Gamma \vdash \texttt{mkvar}\, M_1\, M_2 : \texttt{var}}$$

Table 2.3: Formation rules for IA.

We will sometimes use the ML-like syntactic sugar: “$X := v$” for “$\texttt{assign}\, X\, v$”, “$!X$” for “$\texttt{deref}\, X$”, and “$M; N$” for “$\texttt{seq}\, M\, N$”.

Finitary fragments of Idealized algol

We call ***Finitary Idealized Algol*** the recursion-free sub-fragment of IA defined over finite ground types (*i.e.*, the atomic type $\texttt{exp}$ inhabits the set $\{0..M\}$ for some fixed natural number $M \in \mathbb{N}$).

Definition 2.28 (i^{th} order IA term)**.** A term $\Gamma \vdash M : T$ of finitary Idealized algol is an i^{th}-order term if any sequent $\Gamma' \vdash N : A$ appearing in the typing derivation of M is such that $\operatorname{ord} A \leq i$ and all the variables in Γ' are of order strictly less than i.

The fragment of finitary Idealized Algol consisting of the collection of i^{th}-order terms is denoted IA_i and is called the ***order-i finitary fragment of IA***. If we add the iteration construct defined as

$$\frac{\Gamma \vdash M : \texttt{bool} \qquad \Gamma \vdash N : \texttt{com}}{\Gamma \vdash \texttt{while}\ M\ \texttt{do}\ N : \texttt{com}}\quad \text{where } \forall x \in \Gamma : \operatorname{ord} x < i$$

we obtain the fragments $\mathrm{IA}_i + \mathsf{while}$ for $i \in \mathbb{N}$. Finally $\mathrm{IA}_i + \mathsf{Y}_j$ for $j < i$ denotes the fragment IA_i augmented with a set of fixed-point iterators $\mathsf{Y}_A : (A \to A) \to A$ for every type A of order j at most, whose syntax is defined by the rule:

$$\frac{\Gamma \vdash \lambda x^A.M : A \to A}{\Gamma \vdash \mathsf{Y}_A M : A}\quad \text{where } \forall x \in \Gamma : \operatorname{ord} x < i \text{ and } \operatorname{ord} A \leq j.$$

Operational semantics of IA

To define the operational semantics of IA we proceed slightly differently than for PCF. Instead of giving the semantics for closed terms, we consider terms whose free variables are all of type $\texttt{var}$. A context Γ whose variables are all assigned the type $\texttt{var}$ is called a $\texttt{var}$-context. Terms are “closed” by means of *stores*. A ***store*** is a function mapping free variables of type $\texttt{var}$ to natural numbers. It is called Γ-store just if its domain of definition is precisely the domain of the typing-context Γ. If s is a store then $s \mid x \mapsto n$ denotes the store that maps x to n and acts according to s for other variables.

The set of IA ***canonical forms*** is given by the grammar:

$$V ::= \mathtt{skip} \mid n \mid \lambda x^A.M \mid x \mid \mathtt{mkvar}\ M\ N$$

where n ranges over natural number and x over variable names.

An IA ***program*** is a term together with a Γ-store such that $\Gamma \vdash M : A$. The evaluation semantics is expressed by the judgement form:

$$s, M \Downarrow s', V$$

where s and s' are Γ-stores, V is a canonical form and $\Gamma \vdash V : A$.

The operational semantics for IA is given by the rule of PCF (Table 2.2) together with the rules of Table 2.4 in which the following abbreviation is used:

$$\frac{M_1 \Downarrow V_1 \quad M_2 \Downarrow V_2}{M \Downarrow V} \qquad \text{for} \qquad \frac{s, M_1 \Downarrow s', V_1 \quad s', M_2 \Downarrow s'', V_2}{s, M \Downarrow s'', V} \ .$$

$$\textbf{Sequencing:}\ \frac{M \Downarrow \mathtt{skip} \quad N \Downarrow V}{\mathtt{seq}\ M\ N \Downarrow V}$$

$$\textbf{Variables:}\ \frac{s, N \Downarrow s', n \quad s', M \Downarrow s'', x}{s, \mathtt{assign}\ M\ N \Downarrow (s'' \mid x \mapsto n), \mathtt{skip}} \qquad \frac{s, M \Downarrow s', x}{s, \mathtt{deref}\ M \Downarrow s', s'(x)}$$

$$\textbf{Bad-variables:}\ \frac{N \Downarrow n \quad M \Downarrow \mathtt{mkvar}\ M_1\ M_2 \quad M_1\ n \Downarrow \mathtt{skip}}{\mathtt{assign}\ M\ N \Downarrow \mathtt{skip}} \qquad \frac{N \Downarrow \mathtt{mkvar}\ M_1\ M_2 \quad M_2\ \Downarrow n}{\mathtt{deref}\ M \Downarrow n}$$

$$\textbf{Block:}\ \frac{(s \mid x \mapsto 0), M \Downarrow (s' \mid x \mapsto n), V}{s, \mathtt{new}\ x\ \mathtt{in}\ M \Downarrow s', V}$$

Table 2.4: Big-step operational semantics of IA.

Small-step semantics

The operational semantics of IA can equivalently be defined by means of a small-step semantics: We use reduction rules are of the form $s, M \to s', M'$ where s and s' denote the stores and M and M' denote IA terms. The relation $\to$ is defined by the following rules (We write $M \to M'$ as an abbreviation for $s, M \to s', M'$.):

- *β-reduction:* If $M\ \beta\ M'$ then $M \to M'$;
- *PCF constants:*

$$\begin{array}{rcl} \mathtt{succ}\ n & \to & n+1 \\ \mathtt{pred}\ n+1 & \to & n \\ \mathtt{pred}\ 0 & \to & 0 \\ \mathtt{cond}\ 0\ N_1 N_2 & \to & N_1 \\ \mathtt{cond}\ (n+1)\ N_1 N_2 & \to & N_2 \\ \mathsf{Y} M & \to & M(\mathsf{Y} M)\ ; \end{array}$$

- *IA constants:*

$$
\begin{aligned}
\texttt{seq skip}\ M &\to M \\
s, \texttt{new}\ x\ \texttt{in}\ M &\to (s|x \mapsto 0), M \\
s, \texttt{assign}\ x\ n &\to (s|x \mapsto n), \texttt{skip} \\
s, \texttt{deref}\ x &\to s, s(x) \\
\texttt{assign}\ (\texttt{mkvar} M N)\ n &\to M n \\
\texttt{deref}\ (\texttt{mkvar} M N) &\to N\ ,
\end{aligned}
$$

where n ranges over the natural numbers.

The *redexes*—the expressions occurring in the left-hand side of the reduction rules—can be reduced when occurring as part of a larger expression. The locations where such reduction can occur are defined by means of *evaluation contexts*—expressions containing a hole, denoted by '$-$', indicating a position where a reduction can take place. They are given by the grammar

$$
\begin{aligned}
E[-] \quad ::= \quad & - \mid EN \mid \texttt{succ}\ E \mid \texttt{pred}\ E \mid \texttt{cond}\ E\ N_1\ N_2 \mid \\
& \texttt{seq}\ E\ N \mid \texttt{deref}\ E \mid \texttt{assign}\ E\ n \mid \texttt{assign}\ M\ E \mid \texttt{new}\ x\ \texttt{in}\ E\ .
\end{aligned}
$$

The small-step semantics is then completed with the rule:

$$
\frac{M \to N}{E[M] \to E[N]}\ .
$$

Substitution

The substitution operation naturally extends to IA: it is done inductively on the structure of the term. For the block-variable case this gives:

$$
\begin{aligned}
&(\texttt{new}\ x\ \texttt{in}\ M)\,[N/y] = \texttt{new}\ z\ \texttt{in}\ M\,[z/x]\,[N/y] && \text{if } x \neq y,\ z \text{ fresh;} \\
&(\texttt{new}\ x\ \texttt{in}\ M)\,[N/x] = \texttt{new}\ x\ \texttt{in}\ M\ .
\end{aligned}
$$

For *capture-permitting* substitution, the former equation becomes:

$$
(\texttt{new}\ x\ \texttt{in}\ M)\,\{N/y\} = \texttt{new}\ x\ \texttt{in}\ M\,\{N/y\} \qquad \text{if } x \neq y.
$$

2.2 Higher-Order Grammars and the Safety Restriction

We present the safety restriction in the context of higher-order grammars as it was originally defined [KNU02]. We give a brief introduction to the concept of higher-order grammars. A more detailed introduction on the subject is de Miranda's thesis [dM06].

2.2.1 Higher-order grammars

We consider simple types over a single atom o. Given a set of typed symbols S, the set of ***applicative terms*** generated from S, written $\mathcal{A}(S)$ is defined as the closure of S under the application rule (*i.e.*, if $M : A \to B$ and $N : A$ are in $\mathcal{A}(S)$ then so is $MN : B$).

Definition 2.29. A ***higher-order grammar*** is a tuple $\langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$, where

- Σ is a ranked alphabet (in the sense that each symbol $f \in \Sigma$ has a positive arity written $arity(f)$) of *terminals*;

- $\mathcal{N}$ is a finite set of typed *non-terminals*;

- S is a distinguished ground-type symbol of $\mathcal{N}$, called the start symbol;

- $\mathcal{R}$ is a finite set of production (or rewrite) rules. For each non-terminal $F : (A_1, \ldots, A_n, o) \in \mathcal{N}$ there is (at least) one rule of the form:

$$F z_1 \ldots z_m \rightarrow e$$

where each z_i (called *parameter*) is a variable of type A_i and e is an applicative term of type o generated from the typed symbols in $\Sigma \cup \mathcal{N} \cup \{z_1 : A_1, \ldots, z_m : A_m\}$.

We say that the grammar is *order-n* just in case the order of the highest-order non-terminal is n.

An applicative term generated from the terminals Σ only (without non-terminals), and viewed as a Σ-labelled tree, is called a ***value term***.

Higher-order grammars as generators of term tree languages

From now on we will consider higher-order grammars in which the ranked-alphabet Σ is restricted to terminals of order 1 at most so that each terminal $f \in \Sigma$ has type $o^r \rightarrow o$ where $r \geq 0$ is the arity of f. The idea is that the base type o inhabits the set of trees. An order-0 terminal thus represents a leaf-constructor while an order-1 terminal represents a node-constructor.

A higher-order grammar G determines a tree language denoted $L(G)$ consisting of all the *finite* value terms that can be obtained by normalizing the start symbol S using the reduction relation induced by the rewriting rules of G. This normalization can be done using different reduction strategies, also called *derivation modes*. The main ones are: outside-in (OI), inside-out (IO), and unrestricted. As the names suggest, in the OI derivation mode the outermost redex is reduced first, in IO mode the innermost redex is reduced first, and no particular choice of redex is imposed in unrestricted mode. It can be shown that the OI derivation is sufficient in the sense that every value term obtained from an IO derivation can also be obtained from an OI derivation. The converse however does not hold [Dam82].

Higher-order grammars as word language generators

Higher-order grammars can be used as generators of word languages by imposing the following constraints on the set of terminals Σ:

- Σ contains a special symbol $e : o$,
- all other constants are of type (o, o).

The idea is that the type o represent the type of strings Σ^*, the symbol e marks the end of the word and a constant $f : (o, o)$ represents the operation that appends the letter 'f' as a prefix to a string.

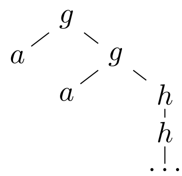
Higher-order grammars as tree generators

In order to generate infinite trees, higher-order grammars are specialized into a device called *recursion scheme*. A ***higher-order recursion scheme***, HORS for short, is a higher-order grammar where the set of rewrite rules is deterministic (*i.e.*, for each non-terminal $F \in \mathcal{N}$ there is exactly one production rule with F on the left-hand side).

A recursion scheme R defines a (potentially infinite) value tree denoted $[\![R]\!]$ obtained by unfolding its rewrite rules *ad infinitum*, replacing formal by actual parameters each time, starting from the start symbol S. Formally, $[\![R]\!]$ is defined as the least upper bound of the *schematological tree grammar* induced by R in the continuous algebra of ranked trees with the appropriate ordering [KNU02, dM06].

Example 2.30. Let G be the following order-2 recursion scheme:

$$\begin{array}{rcl} S & \rightarrow & H\,a \\ H\,z & \rightarrow & F\,(g\,z) \\ F\,\phi & \rightarrow & \phi\,(\phi\,(F\,h)) \end{array}$$



with non-terminals $S : o$, $F : ((o, o), o)$, $H : (o, o)$ and terminals g, h, a of arity $2, 1, 0$ respectively. Then the tree generated by G is defined by the infinite term $g\,a\,(g\,a\,(h\,(h\,(h\,\cdots))))$ pictured on the right.

2.2.2 The safety restriction

Safety is a syntactic restriction for higher-order grammars introduced by Knapik et al. in order to study the Monadic Second Order (MSO) theory of infinite trees generated by higher-order pushdown automata [KNU02]. The safety restriction has appeared under different forms in the literature. The first formulation, due to Damm, appeared under the name *restriction of derived types* [Dam82]. De Miranda's thesis contains a comparison of the two formulations [dM06]. The presentation given here follows that of Knapik et al. [KNU02].

Type homogeneity

We say that a type is ***homogeneous*** if it is o or if it is $(A_1, \cdots, A_n, o)$ with the condition that $\mathrm{ord}\,A_1 \geq \mathrm{ord}\,A_2 \geq \cdots \geq \mathrm{ord}\,A_n$ and each $A_1, \ldots, A_n$ is homogeneous [KNU02].

Notation 2.31 (Type partitioning) Suppose that $\overline{A_1}, \overline{A_2}, \ldots, \overline{A_n}$ are n lists of types, where A_{ij} denotes the j^{th} type in the list $\overline{A_i}$ and l_i the size of $\overline{A_i}$. We introduces the following notation that partitions the A_{ij}s according to their order:

$$A \;=\; (\overline{A_1}\,|\,\cdots\,|\,\overline{A_r}\,|\,o)$$

to mean that

- A is the type $(A_{11}, A_{12}, \cdots, A_{1l_1}, A_{21}, \cdots, A_{2l_2}, \cdots A_{n1}, \cdots, A_{nl_n}, o)$,
- $\forall i : \forall u, v \in A_i : \mathrm{ord}\,u = \mathrm{ord}\,v$,
- $\forall i, j.\forall u \in A_i.\forall v \in A_j.i < j \implies \mathrm{ord}\,u > \mathrm{ord}\,v$.

So in particular A is homogeneous. If further we have $B = (\overline{B_1}\,|\,\cdots\,|\,\overline{B_m}\,|\,o)$ then we use the notation $(\overline{A_1}\,|\,\cdots\,|\,\overline{A_n}\,|\,B)$ as an abbreviation for $(\overline{A_1}\,|\,\cdots\,|\,\overline{A_n}\,|\,\overline{B_1}\,|\,\cdots\,|\,\overline{B_m}\,|\,o)$.

Definition

Definition 2.32 (Safe grammar)**.** (All types are assumed to be homogeneous.) A term of order $k > 0$ is *unsafe* if it contains an occurrence of a parameter of order strictly less than k, otherwise the term is *safe*. An occurrence of an unsafe term t as a subexpression of a term t' is *safe* if it is in the context $\cdots (ts) \cdots$, otherwise the occurrence is *unsafe*. A grammar is ***safe*** if no unsafe term has an unsafe occurrence at a right-hand side of any production.

This definition is a bit opaque and does not seem to make a lot of sense at first. One can reformulate it in a slightly clearer way: A higher-order grammar G whose non-terminals are of homogeneous type is *unsafe* if and only if there is a rewrite rule $F z_1 \ldots z_m \rightarrow e$ where e contains a subterm that:

1. occurs in *operand* position in e,
2. contains a parameter of order strictly less than its order.

(By "operand position" we mean "in the second position of some occurrence of the implicit application operator of the lambda calculus".) A grammar is *safe* if it is not unsafe.

Example 2.33 ([KNU02])**.** Let $f : (o, o, o)$, $g, h : (o, o)$ and $a, b : o$ be Σ constants. The grammar of level 3 with non-terminals $S : o$ and $F : ((o, o), o, o, o)$ and production rules:

$$\begin{array}{rcl} S & \rightarrow & Fgab \\ F\varphi xy & \rightarrow & f(F(F\varphi x)y(hy))(f(\varphi x)y) \end{array}$$

is not safe because the subterm $F\varphi x$, in the right-hand side expression of the second rule, is of type (o, o), contains a ground-type variable and occurs at an operand position.

On the other hand, the following production rules are safe:

$$\begin{array}{rcl} S & \rightarrow & G(ga)b \\ Gzy & \rightarrow & f(G(Gzy)(hy))(fzy) \ . \end{array}$$

It can be shown [KNU02] that these rules are equivalent to the ones given above in the sense that the induced recursion schemes generate the same infinite tree.

Example 2.34. Let $F : ((o, o), o, o, o)$, $G : (o, o)$ and $H : ((o, o), o)$ be non-terminals and $f : (o, o, o)$ be a terminal. Then the following rewrite rules are unsafe. (The unsafe occurrences of unsafe subterms are underlined.):

$$\begin{array}{rcl} G\,x & \rightarrow & H\,\underline{(f\,x)} \\ F\,z\,x\,y & \rightarrow & f\,(F\,\underline{(F\,z\,y)}\,y\,(z\,x))\,x \ . \end{array}$$

Example 2.35. The order-2 grammar defined in Example 2.30 is unsafe.

2.2.3 Automata-theoretic Characterization

Although very technical, the safety restriction for higher-order recursion schemes has an appealing machine characterization. Knapik, Niwiński and Urzyczyn showed that for generating infinite ranked trees, safe higher-order recursion schemes are as expressive as *higher-order pushdown automata (PDA)* [KNU02].

A ***pushdown automaton*** (PDA) is an infinite-state transition system that can manipulate the content of a stack when performing a transition. Higher-order pushdown automata were introduced as a generalization of PDA [Mas76]. Instead of manipulating a simple stack, a higher-order PDA manipulates iterated stacks. An order-1 PDA is an ordinary PDA, an order-2 PDA manipulates order-2 stacks which are stacks of order-1 stacks. In addition to the usual push and pop transitions of a PDA, an order-2 PDA has order-2 variants: a $push_2$ operation that duplicates the top order-1 stack, and a pop_2 that pops the entire top order-1 stack. This definition generalizes to any order $n \in \mathbb{N}$.

Theorem 2.36 (Knapik, Niwiński and Urzyczyn, [KNU02])**.** *Let L be a Σ-labelled term tree language. L is the language of a safe order-n grammar (using the OI derivation) if and only if it is accepted by an order-n pushdown automaton.*

So in particular, a (potentially) infinite tree t is generated by a safe order-n recursion scheme if and only if it is accepted by an order-n pushdown automaton.

A similar characterization has subsequently been obtained for unrestricted grammars: Hague, Murawski, Ong and Serre have introduced a new kind of pushdown automata called *collapsible pushdown automata* (CPDA) and showed their equivalence with unrestricted higher-order grammars. The internal structure manipulated by a CPDA is a stack in which every symbol has a link pointing to some other substacks situated below it. There is an additional stack-operation called *collapse* whose effect is to replace the content of the stack by the sub-stack indicated by the link attached to the top symbol of the stack.

Theorem 2.37 (Hague, Murawski, Ong and Serre, [HMOS08])**.** *A potentially infinite (ranked) tree t is generated by an order-n recursion scheme if and only if it is accepted by an order-n collapsible pushdown automaton.*

We have defined higher-order grammars as generators of word languages and trees. Thanks to the machine characterization, it is possible to define the notion of graph generated by a higher-order grammars: the graph generated by a grammar is defined as the configuration graph of the corresponding collapsible higher-order pushdown automaton. In particular, the graph generated by a safe grammar is the configuration graph of the corresponding higher-order PDA.

2.2.4 Expressivity

Higher-order PDA/grammars can be used as generating device for word-languages, trees, or graphs, thus inducing strict infinite hierarchies as the order of the PDA varies. For word-languages this is known as the Maslov hierarchy: orders 0, 1 and 2 correspond respectively to the regular, context-free and indexed languages. For trees, orders 0, 1 and 2 are respectively the regular, algebraic and hyperalgebraic trees.

2.2.5 Is safety a genuine restriction?

The implications that the safety constraint has on the expressivity of higher-order grammars are not completely understood. A partial answer has been given for word languages: Aehlig, de Miranda and Ong showed that up to order 2, there is no intrinsically unsafe word language [AdMO05b]: any word language generated by an unsafe order-2 grammar can also be generated by some (potentially non-deterministic) order-2 safe grammar. For trees, Urzyczyn conjectured [dM06] that safety constrains expressivity. He even proposed a tree—known as Urzyczyn's tree—generated by an unsafe order-2 recursion scheme that he conjectured to not be generated by any safe order-2 recursion scheme. At the time of this writing, this still remains a conjecture.

A similar question can be asked from a verification point of view: *Are the structures generated by safe higher-order grammars easier to verify that those generated by unrestricted grammars?* The reason why the safety constraint was introduced in the first place was precisely to be able to show that the generated trees have decidable Monadic Second Order (MSO) theories [KNU02]. In fact, it was subsequently shown that this result also holds in the general unrestricted case [Ong06a]:

Theorem 2.38 (Ong, 2006)**.** *The modal mu-calculus model checking problem for trees generated by order-n recursion schemes is n-EXPTIME complete for each $n \geq 0$.*

This result implies that these trees have decidable MSO theories since the two logics are equi-expressive over trees. The proof of this theorem relies on a game-semantic argument based on the theory of traversals (that will be presented in Chapter 4) which radically differs from the argument used by Knapik et al. for the case of safe grammars [KNU02]. A generalization of Theorem 2.38 for graphs was later obtained by Hague et al. [HMOS08]:

Theorem 2.39 (Hague et al., 2008)**.** *For each $n \geq 0$, the modal mu-calculus model checking problem for configuration graphs of order-n collapsible pushdown systems is n-EXPTIME complete.*

For graphs, the MSO logic is strictly more expressive than the modal mu-calculus. In the same paper it is shown that MSO theories of collapsible pushdown graphs are undecidable while those of pushdown graphs are decidable [HMOS08]. Hence from a verification point of view, safety can indeed be considered as a genuine constraint.

2.2.6 Higher-order grammars and the simply-typed lambda calculus

There is a natural correspondence between higher-order grammars and the simply-typed lambda calculus: deterministic higher-order grammars (*i.e.*, recursion schemes) are essentially closed simply-typed lambda-terms of ground type extended with mutual recursion and generated from the terminal symbols Σ of the grammar. A similar correspondence holds between (possibly non-deterministic) higher-order grammars and the simply-typed lambda calculus extended with a non-deterministic branching operator. We now show how this correspondence works in the deterministic case.

Let $\Lambda^{mut}_{\rightarrow}(\Sigma)$ denote the simply-typed lambda calculus extended with mutual recursion and generated from the set of typed constants Σ. The syntax of the mutual recursion operator is given by the typing-rule

$$(\mathsf{Y_{mut}})\frac{\Gamma \vdash_\Sigma M_1 : A \rightarrow A_1 \qquad \Gamma \vdash_\Sigma M_q : A \rightarrow A_q}{\Gamma \vdash_\Sigma \mathsf{Y_{mut}}(M_1, \ldots, M_q) : A_1}\ A = A_1 \times \ldots \times A_q, q \geq 0$$

whose semantics is given by

$$\begin{aligned}\mathsf{Y_{mut}}(M_1, \ldots, M_q) &\rightarrow \pi_1(\mathsf{Y}\langle M_1 \ldots M_q\rangle)\ ,\\ \mathsf{Y}\langle M_1, \ldots, M_q\rangle &\rightarrow \langle M_1(\mathsf{Y}\langle M_1, \ldots, M_q\rangle), \ldots, M_q(\mathsf{Y}\langle M_1, \ldots, M_q\rangle)\rangle\ ,\end{aligned}$$

where π_1 denotes the first projection for q-tuples. (The operator Y denotes the usual Y combinator of PCF extended to product types.)

Let $R = \langle \Sigma, \mathcal{N}, \mathcal{R}, F_0\rangle$ be a higher-order recursion scheme with $\mathcal{N} = \{F_0, \ldots, F_q\}$ and $\mathcal{R} = \{F_i\ x_1 \ldots x_n \rightarrow e_i \mid 0 \leq i \leq q\}$ for some $q \geq 0$. We define the closed $\Lambda^{mut}_{\rightarrow}(\Sigma)$-term $\mathsf{HORStoLmd}(R)$ as follows:

$$\begin{aligned}\mathsf{HORStoLmd}(R) &\equiv \mathsf{Y_{mut}}(\widetilde{F_0},\ \ldots,\ \widetilde{F_q})\\ \widetilde{F_i} &\equiv \lambda F_0 \ldots F_q x_1 \ldots x_n.e_i \qquad \text{for } 0 \leq i \leq q\ .\end{aligned}$$

Conversely, every $\Lambda^{mut}_{\rightarrow}(\Sigma)$-term can be reformulated as a higher-order recursion scheme. The algorithm $\mathsf{LmdToHORS}$ of Table 2.5, described in an ML-like pseudo-code, takes a closed $\Lambda^{mut}_{\rightarrow}(\Sigma)$-term and returns the corresponding higher-order recursion scheme. It proceeds inductively over the syntax of the term. The local variables $\mathcal{N}$ and $\mathcal{R}$ are used to accumulate respectively the non-terminals and rewrite rules of the recursion scheme being built. The auxiliary function $\mathsf{createRules}$ is responsible for creating the rules for a given open lambda-term; it adds them to the set $\mathcal{R}$ and returns and applicative term from $\mathcal{A}(\mathcal{N} \cup \Sigma)$ corresponding to the input lambda-term. (The symbol '@' denotes the data-constructor used to build lambda-term applications.)

It is straightforward to check that for every higher-order recursion scheme R the recursion scheme $\mathsf{LmdToHORS}(\mathsf{HORSToLmd}(R))$ is the same as R (up to renaming of the non-terminals and rule parameters).

Example 2.40. Let R denote the recursion scheme of Example 2.30. We have:

$$\begin{aligned}\mathsf{HORSToLmd}(R) &\equiv \mathsf{Y_{mut}}(\widetilde{S}, \widetilde{H}, \widetilde{F})\\ \text{where } \widetilde{S} &\equiv \lambda SHF.H\, a\\ \widetilde{H} &\equiv \lambda SHFz.F\,(g\, z)\\ \widetilde{F} &\equiv \lambda SHF\phi.\phi\,(\phi\,(F\, h))\ .\end{aligned}$$

Converting this term back to a HOG gives $\mathsf{LmdToHORS}(\mathsf{HORSToLmd}(R)) = \langle \Sigma, \mathcal{N}, \mathcal{R}, S\rangle$ where $\mathcal{N} = \{S : o, \widehat{F_1} : o, \widehat{F_2} : (o, o), \widehat{F_3} : ((o, o), o)\}$ and

$$\mathcal{R} = \{S \rightarrow \widehat{F_1}, \quad \widehat{F_1} \rightarrow \widehat{F_2}\, a, \quad \widehat{F_2}\, z \rightarrow \widehat{F_3}\,(g\, z), \quad \widehat{F_3}\, \psi \rightarrow \psi(\psi(\widehat{F_3}\, h))\}\ .$$

Input: A closed $\Lambda^{mut}_{\rightarrow}(\Sigma)$-term $\vdash_\Sigma M : T$.
Output: A higher-order recursion scheme $\langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$.

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{let\ LmdToHORS}(\vdash_\Sigma M : T)} \\
\multicolumn{3}{l}{\quad \mathsf{let\ createRules} : \Lambda^{mut}_{\rightarrow}(\Sigma) \rightarrow \mathcal{A}(\mathcal{N} \cup \Sigma) = \mathsf{fun}} \\
\quad\quad \mid \Gamma \vdash_\Sigma \alpha : T \quad \text{with } \alpha \in \Gamma \cup \Sigma & \rightarrow & \alpha \\
\quad\quad \mid \Gamma \vdash_\Sigma M N : B & \rightarrow & \mathsf{createRules}(\Gamma \vdash_\Sigma M : A \rightarrow B) \\
 & & @\mathsf{createRules}(\Gamma \vdash_\Sigma N : A) \\
\quad\quad \mid \overline{x} : \overline{A} \vdash_\Sigma \lambda y_1^{B_1} \ldots y_k^{B_k}.M : (\overline{B}, o) & \rightarrow & \mathsf{let}\ \Gamma = \overline{x} : \overline{A}, y_1 : B_1, \ldots, y_n : B_n \\
\quad\quad\ \ \textit{where } M \textit{ is not an abstraction,} & & \textit{for some fresh names } y_{k+1} \ldots y_n\ \mathsf{in} \\
\quad\quad\ \ \overline{B} = (B_1 \ldots B_n),\ \textit{and } 1 \leq k \leq n, & & \mathsf{let}\ e = \mathsf{createRules}(\Gamma \vdash_\Sigma M \lceil y_{k+1} \rceil \ldots \lceil y_n \rceil : o) \\
 & & \mathsf{and}\ F \text{ be a fresh non-terminal name } \mathsf{in} \\
 & & \mathcal{R} \leftarrow \text{“}F\ \overline{x}\ \overline{y} \rightarrow e\text{”} :: \mathcal{R} \\
 & & \mathcal{N} \leftarrow \text{“}F : (\overline{A}, \overline{B}, o)\text{”} :: \mathcal{N} \\
 & & \text{“}F\ \overline{x}\text{”} \\
\quad\quad \mid \overline{x} : \overline{A} \vdash_\Sigma \mathsf{Y_{mut}}(M_1, \ldots, M_q) : B_1 & \rightarrow & \mathsf{for\ i = 1\ ..\ q\ do} \\
\quad\quad\ \ \textit{where } M_i : B_i \textit{ for } i \in \{1..q\}, & & \quad \mathsf{createRules}(\overline{x} : \overline{A} \vdash_\Sigma M_i : B_i) \\
 & & \quad \mathsf{let}\ \text{“}F\,\overline{x}\,f_1 \ldots f_q\,\overline{y} \rightarrow e\text{”} \leftarrow \textit{hd}\ \mathcal{R}\ \mathsf{in} \\
 & & \quad \mathcal{R} \leftarrow \text{“}\widehat{F_i}\ \overline{x}\ \overline{y} \rightarrow e[\widehat{F_1}\overline{x}/f_1] \cdots [\widehat{F_q}\overline{x}/f_q]\text{”} \\
 & & \quad\quad\quad :: \textit{tail}\ \mathcal{R} \\
 & & \quad \mathcal{N} \leftarrow \text{“}\widehat{F_i} : (\overline{A}, B_i)\text{”} :: \textit{tail}\ \mathcal{N} \\
 & & \mathsf{done} \\
 & & \text{“}\widehat{F_1}\ \overline{x}\text{”} \\
\multicolumn{3}{l}{\quad\ \ \mathsf{in}} \\
\multicolumn{3}{l}{\quad\ \ \mathcal{N}, \mathcal{R} \leftarrow [], []} \\
\multicolumn{3}{l}{\quad\ \ \textit{appterm} \leftarrow \mathsf{createRules}(\vdash_\Sigma M : T)} \\
\multicolumn{3}{l}{\quad\ \ \langle \Sigma, \text{“}S : o\text{”} :: \mathcal{N}, \text{“}S \ \rightarrow \textit{appterm}\text{”} :: \mathcal{R}, S \rangle}
\end{array}
$$

Table 2.5: Algorithm LmdToHORS converting a mutually recursive lambda-term into a higher-order recursion scheme.

The following intermediary rules are created during the execution of the algorithm:

$$F_1\, S\, H\, F \rightarrow H\, a, \quad F_2\, S\, H\, F\, z \rightarrow F\,(g\, z), \quad F_3\, S\, H\, F \psi \rightarrow \psi(\psi(F\, h)) \enspace ,$$

where $F_1 : (o,(o,o),((o,o),o),o)$, $F_2 : (o,(o,o),((o,o),o),o,o)$, $F_3 : (o,(o,o),((o,o),o),(o,o),o)$.

2.3 Game Semantics

Game semantics is a very powerful paradigm for giving models of programming languages. It was the first kind of semantics able to provide a *fully abstract model* of the language PCF, a result which was subsequently extended to other languages. In a nutshell, the term "full abstraction" means that the model provides a faithful mathematical characterization of the language. A natural way to give a semantic account of a language consists therefore in giving a game-semantic characterization of it. A question that we will try to answer in this thesis is: How does a syntactic restriction such as *safety* impact on the on the game model of a language? A substantial part of this thesis is devoted to this question (Chapter 4 and 6).

This chapter introduces the basic notions of game semantics including the categorical interpretation, the game interpretation of PCF and IA, and the full abstraction results. It concludes by giving a brief summary of some important results in *algorithmic game semantics*. For an introduction, we recommend the tutorial by Samson Abramsky [AM98b] on which this chapter is based. Many details and proofs will be omitted; we refer the reader to other literature [HO00, AMJ94] for a complete account. The reader familiar with game semantics may very well consider skipping this chapter altogether as all the definitions and notations introduced here are standard.

2.3.1 Historical remarks

We give an outline of the history of game semantics. Cardone and Hindley gave a more detailed survey [CH06].

Logic

Game semantics finds its origin in various works [Lor61, BC82, Bla92, Joy77]. Paul Lorenzen introduced a game semantics for logic in the 1950s to study intuitionistic logic [Lor61] where the notion of logical truth is modeled using game-theoretic concepts such as the existence of a winning strategy. Four decades later, this approach was used by Andreas Blass [Bla92] to establish a connection with Girard's linear logic. Joyal [Joy77] later presented his "combinatorial" calculus of strategies, establishing the first categorical account of two-player games. In the 1990s, Samson Abramsky and Radha Jagadeesan [AJ92] on one hand, Martin Hyland and Luke Ong [HO93] on the other hand, used game semantics to prove full completeness of Multiplicative Linear Logic (MLL).

Models of programming languages

Subsequently, game semantics emerged as a new paradigm for the study of formal models for programming languages. Three different independent research groups: Samson Abramsky, Radhakrishnan Jagadeesan and Pasquale Malacaria [AMJ94]; Martin Hyland and Luke Ong [HO00]; and Nickau [Nic94] introduced a new kind of model based on game semantics in order to solve a long standing problem in the semanticists community: finding a fully abstract model for PCF.

Many approaches were used to define models for programming languages before the introduction of game models. Among the successful ones were the:

- *operational semantics*: The meaning of a program is defined by describing the behaviour of a machine executing it. This is formally done by means of a state transition system;

- *axiomatic semantics*: The behaviour of the program is defined by means of axioms. This kind of semantics lends itself well to proving correctness of the program by static analysis of the program code;
- *denotational semantics*: Programs are mapped to mathematical objects with good properties (such as compositionality). This mapping is done by structural induction on the syntax of the program.

In *game semantics*, the idea is to model the program as a game played by two protagonists: the Opponent, representing the environment, and the Proponent, representing the program. The meaning of the program is then modeled by a strategy for the Proponent.

The problem of full abstraction for PCF

The problem of the Full Abstraction for PCF goes back to the 1970s. Scott constructed a model of PCF based on domain theory [Sco93] which gives a sound interpretation of observational equivalence: if two terms have the same domain theoretic interpretation then they are observationally equivalent. However the converse is not true: There exist two PCF terms which are observationally equivalent but have different domain theoretic denotations—we say that the model is not *fully abstract*.

The reason why the domain theoretic model is not fully abstract lies in the fact that the *parallel-or* operator defined by the following truth table

p-or	$\perp$	tt	ff
$\perp$	$\perp$	tt	$\perp$
tt	tt	tt	tt
ff	$\perp$	tt	ff

is not definable by any PCF term. Indeed, it is possible to define two different PCF terms that have the same behaviour except when applied to a term computing p-or. In the domain-theoretic model these two terms have different denotation, but they are equivalent since p-or is not definable in PCF. Hence the model is not fully abstract.

One solution to the problem is to "patch" PCF by adding the p-or operator. The resulting language "PCF+p-or" was shown to be fully-abstracted by Scott domain theoretic model [Plo77]. The language that we are now dealing with, however, is strictly more powerful than PCF—it allows parallel execution of commands whereas PCF only permits sequential execution.

Another approach involves the elimination of the undefinable elements (like p-or) by strengthening the conditions on the functions used in the model. This approach was followed by Berry who gave a model based on stable functions [Ber78, Ber79], a class of functions smaller than the class of strict and continuous functions. Unfortunately this approach did not succeed.

Fully abstract models for PCF were found at the same time and independently by three research teams: Abramsky, Jagadeesan and Malacaria [AMJ94], Hyland and Ong [HO00] and Nickau [Nic94]. These three approaches are all based on game semantics.The game-semantic approach has subsequently been adapted to other varieties of programming paradigms leading to fully abstract models of languages featuring stores (Idealized Algol), call-by-value [HY99, AM98a] and call-by-name, general references [AHM98], polymorphism [AJ05], control features (call with current continuation), non determinism, concurrency, etc.

2.3.2 Definitions

We now introduce formally the notion of game that we will use in later sections to model programming languages. We consider a two-player game. The players are named O for ***Opponent*** and P for ***Proponent***. The game played by these two players is constrained by an *arena*. The arena defines the possible moves of the game. By analogy with board games, the arena represents the board together with rules indicating which are the legal moves for each player. The analogy with board game stops here. Instead we regard our game as a dialog between two players unfolding as follows: The Opponent interviews the Proponent; P's goal is to answer the initial question asked by O. P can also ask intermediary questions to O in order to request more precision about O's initial question; O can subsequently ask further questions to P. We thus distinguish two kinds of moves in our games: the questions and the answers. This process induces a flow of questions and answers between O and P which can possibly last forever. In game semantics, attention is given to the study of this flow of questions and answers; the notion of 'winning a game' or 'winner of the game' is not a concern.

2.3.2.1 Arenas

The arena defines the bases of the game for the players. It is formally given by a directed acyclic graph (DAG) whose internal nodes correspond to question moves and leaves correspond to answer moves.

Definition 2.41 (Arena)**.** An ***arena*** is a structure $\langle M, \lambda, \vdash\rangle$ where:

- M is the set of possible moves;
- $\lambda : M \to \{O, P\} \times \{Q, A\}$ is a labelling function specifying which are the question and answer moves, and which moves can be played by O and P. Formally, it is given by a pair of functions $\lambda^{OP} : M \to \{O, P\}$ and $\lambda^{QA} : M \to \{Q, A\}$ such that λ is the pairing $\langle\lambda^{OP}, \lambda^{QA}\rangle$. An element m of M is an O-move if $\lambda^{OP}(m) = O$ and a P-move otherwise; it is a question if $\lambda^{QA}(m) = Q$ and an answer otherwise.
- $\vdash$ is an ***enabling relation*** on $M \times M$ such that $(M, \vdash)$ is a directed acyclic graph (DAG) satisfying the following conditions:
  - (e1) The roots are O-questions: For every DAG's root r, $\lambda(r) = OQ$;
  - (e2) Internal nodes of the DAG are questions: $m \vdash n \implies \lambda^{QA}(m) = Q$ (thus answers moves are necessarily leaves);
  - (e3) A player move can only enable moves played by the other player: $m \vdash n \implies \lambda^{OP}(m) \neq \lambda^{OP}(n)$.

We abbreviate the set $\{O, P\} \times \{Q, A\}$ as $\{OQ, OA, PQ, PA\}$. $\overline{\lambda}$ denotes the labelling function obtained by swapping the role of the Opponent and Proponent in λ:

$$\begin{aligned} \overline{\lambda(m)} &= OQ \iff \lambda(m) = PQ \\ \text{and } \overline{\lambda(m)} &= OA \iff \lambda(m) = PA \ . \end{aligned}$$

The roots of the DAG $(M, \vdash)$ are called the ***initial moves***.

The simplest possible arena is the one with an empty set of moves; it is written $\mathbf{1}$.

Example 2.42 (The flat arena)**.** Let A be any countable set. The flat arena over A is defined as the arena $\langle M, \lambda, \vdash\rangle$ such that M has one move q with $\lambda(q) = OQ$ and for each element in A, there is a corresponding move a_i in M with $\lambda(a_i) = PA$ for some $i \in \mathbb{N}$. The enabling relation $\vdash$ is defined to be $\{q \vdash a_i \mid i \in \mathbb{N}\}$. This arena is represented by the tree $\begin{array}{c} q \\ / \mid \backslash \\ a_0 \ a_1 \ \cdots \end{array}$ whose nodes represent the moves and edges represent the enabling relation. In the rest of this thesis we will just write $\mathbb{N}$ to mean the flat arena over $\mathbb{N}$:

$$\begin{array}{c} q \\ / \ / \ \backslash \ \backslash \\ 0 \quad 1 \quad 2 \ \cdots \end{array}$$

Definition 2.43 (Justified sequence of moves)**.** A justified sequence is a sequence of moves s together with an associated sequence of pointers. Any move m in the sequence that is not initial has a pointer that points to a previous move n that enables it (*i.e.*, $n \vdash m$).

(Formally we can regard a justified sequence as a sequence of pairs, each pair encoding an element of the sequence together with an index indicating the position where the element points to.)

Since initial moves are all O-moves, the first move of a justified sequence is necessarily an O-move.

Convention 2.44 Justification pointers are graphically represented with arrows as follows:

$$q^4\ q^3\ q^2\ q^3\ q^2\ q^1 \quad .$$

We will sometimes omit the justification pointers altogether if they do not play any role in the argument.

Notation 2.45 We write $s \cdot t$, or just $s\,t$, to denote the justified sequence obtained by concatenating s and t. The empty sequence is written ϵ. Given a justified sequence $s = m_1 \cdot m_2 \ldots m_n$ (where pointers are not represented) we write $s_{\leqslant m_i}$ for $m_1 \cdot m_2 \ldots m_i$ (the prefix sequence of s up to the move m_i); and $s_{<m_i}$ for $m_1 \cdot m_2 \ldots m_{i-1}$.

Definition 2.46 (Hereditary projection)**.** Let s be a justified sequence of moves. We say that a move m_0 occurring in s is hereditarily justified by a move n occurring in s if there exist moves $m_1, \ldots, m_q$ occurring in s for $q \geq 0$ such that n justifies m_q and m_k justifies m_{k-1} for $1 \leq k \leq q$.

Suppose that n is an occurrence of a move in the sequence s then $s \upharpoonright n$ denotes the subsequence of s consisting of the moves hereditarily justified by n. If I is a set of initial moves then $s \upharpoonright I$ denotes the subsequence of s consisting of the moves hereditarily justified by moves in I.

Justified sequences of moves will be used to record the history of all the moves that have been played so far in the (yet to be defined) game. Two particular subsequences called the *P-view* and the *O-view* are of interest. These subsequences correspond to restricted views that each player has of the history of the game in a given position.

Definition 2.47 (View)**.** Given a justified sequence of moves s, the ***Proponent view*** (P-view) written $\ulcorner s \urcorner$ is defined by induction as follows:

$$
\begin{aligned}
\ulcorner \epsilon \urcorner &= \epsilon, && \\
\ulcorner s \cdot m \urcorner &= \ulcorner s \urcorner \cdot\ m && \text{if } m \text{ is a P-move,} \\
\ulcorner s \cdot m \urcorner &= m && \text{if } m \text{ is initial (O-move) ,} \\
\ulcorner s \cdot m \cdot t \cdot n \urcorner &= \ulcorner s \urcorner \cdot m \cdot n && \text{if } n \text{ is a non initial O-move .}
\end{aligned}
$$

The ***O-view*** $\llcorner s \lrcorner$ is defined similarly:

$$
\begin{aligned}
\llcorner \epsilon \lrcorner &= \epsilon, && \\
\llcorner s \cdot m \lrcorner &= \llcorner s \lrcorner \cdot\ m && \text{if } m \text{ is a O-move,} \\
\llcorner s \cdot m \cdot t \cdot n \lrcorner &= \llcorner s \lrcorner \cdot m \cdot n && \text{if } n \text{ is a P-move .}
\end{aligned}
$$

2.3.2.2 Games

Only certain kinds of justified sequences will be of interest in our games. We call *legal position* any justified sequence that satisfies two conditions: alternation and visibility. Alternation says that players O and P play alternatively. Visibility expresses that each non-initial move is justified by a move situated in the local context at that point. Formally:

Definition 2.48 (Legal position)**.** A legal position is a justified sequence of moves s respecting the following constraints:

- *Alternation*: For every subsequence $m \cdot n$ of s, $\lambda^{OP}(m) \neq \lambda^{OP}(n)$.
- *Visibility*: For every subsequence $t \cdot m$ of s where m is not initial, if m is a P-move then m points to a move occurring in $\ulcorner s \urcorner$; and if m is a O-move then m points to a move occurring in $\llcorner s \lrcorner$.

The set of legal positions of an arena A is denoted by L_A.

Definition 2.49 (Game)**.** A game is a structure $\langle M, \lambda, \vdash, P\rangle$ such that

- $\langle M, \lambda, \vdash\rangle$ is an arena;
- P, called the set of *valid positions*, is:
  - a non-empty prefix closed subset of the set of legal positions,
  - closed by initial hereditary projection: If s is a valid position then for every set I of occurrences of initial moves in s, $s \upharpoonright I$ is also a valid position.

The empty arena **1** together with the empty set of valid positions defines the simplest possible game, also denoted **1**.

Example 2.50. Consider the flat arena $\mathbb{N}$. The set of valid positions $P = \{\epsilon, q\} \cup \{q \cdot a_i \mid i \in \mathbb{N}\}$ defines a game on the arena $\mathbb{N}$.

2.3.2.3 Constructions on games

We now present basic constructions on games.

Consider the two functions $f : A \to C$ and $g : B \to C$, we write $[f, g]$ to denote the pairing of f and g defined on the direct sum $A + B$. Given a game A with a set of moves M_A, we use the projection operator $s \upharpoonright A$ to denote the subsequence of s consisting of all moves in M_A. Although this notation conflicts with the hereditary projection operator, it should not cause any confusion.

Tensor product Given two games A and B the tensor product $A \otimes B$ is defined as:

$$\begin{aligned}
M_{A\otimes B} &= M_A + M_B \\
\lambda_{A\otimes B} &= [\lambda_A, \lambda_B] \\
\vdash_{A\otimes B} &= \vdash_A \ \cup \ \vdash_B \\
P_{A\otimes B} &= \{s \in L_{A\otimes B} | s \upharpoonright A \in P_A \wedge s \upharpoonright B \in P_B\} \ .
\end{aligned}$$

In particular, n is initial in $A \otimes B$ if and only if n is initial in A or B. And $m \vdash_{A\otimes B} n$ holds if and only if $m \vdash_A n$ or $m \vdash_B n$ holds.

Function space The game $A \multimap B$ is defined as follows:

$$\begin{aligned}
M_{A\multimap B} &= M_A + M_B \\
\lambda_{A\multimap B} &= [\overline{\lambda_A}, \lambda_B] \\
\vdash_{A\multimap B} &= \vdash_A \ \cup \ \vdash_B \ \cup \{(m, n) \mid m \text{ initial in } B \wedge n \text{ initial in } A\} \\
P_{A\otimes B} &= \{s \in L_{A\otimes B} | s \upharpoonright A \in P_A \wedge s \upharpoonright B \in P_B\} \ .
\end{aligned}$$

Cartesian product The game $A \times B$ is defined as follows:

$$\begin{array}{rcl} M_{A\times B} & = & M_A + M_B \\ \lambda_{A\times B} & = & [\lambda_A, \lambda_B] \\ \vdash_{A\times B} & = & \vdash_A \ \cup \ \vdash_B \\ P_{A\times B} & = & \{s \in L_{A\otimes B} | s \upharpoonright A \in P_A \wedge s \ \upharpoonright B = \epsilon\} \\ & & \cup \{s \in L_{A\otimes B} | s \upharpoonright A \in P_B \wedge s \ \upharpoonright A = \epsilon\} \ . \end{array}$$

Note that a play of the game $A \times B$ is either a play of A or a play of B, whereas a play of the game $A \otimes B$ may be an interleaving of plays of A and B.

2.3.2.4 Representation of plays

Plays of the game are usually represented in a table diagram. The columns of the table correspond to the different components of the arena and every row corresponds to a move in the play. The first row always represents an O-move, this is because O is the only player who can open a game (since roots of the arena are O-moves).

For example the play $q\, q\, 8\, 9$ on the game $\mathbb{N} \multimap \mathbb{N}$ is represented by the following diagram:

$$\begin{array}{cccc} \mathbb{N} & \multimap & \mathbb{N} & \\ & & q & O \\ q & & & P \\ 8 & & & O \\ & & 9 & P \end{array}$$

We sometimes also represent the justification pointers on the diagram.

2.3.2.5 Strategy

During the game, a player may face several choices when it is his turn to play. A *strategy* is a guide telling the player which move to make when the game is in a given position.

Definition 2.51. A ***strategy*** for player P on a given game $\langle M, \lambda, \vdash, P\rangle$ is a non-empty set of even-length positions from P such that:

1. if $sab \in \sigma$ then $s \in \sigma$ (*no unreachable position*);
2. if $sab, sac \in \sigma$ then $b = c$ and b has the same justifier as c (*determinacy*).

(Alternatively, a strategy can be viewed as a partial function mapping odd-length legal positions to P-moves plus pointers.)

The idea is that the presence of the even-length sequence sab in σ tells the player P that whenever the game is in position s and player O plays the move a then it must respond by playing the move b. The first condition ensures that the strategy σ only considers positions that the strategy itself could have led to in a previous move. The second condition in the definition requires that this choice of move is deterministic (*i.e.*, there is a function f from the set of odd length position to the set of moves M such that $f(sa) = b$) as well as the choice of its pointer.

For every game A, the smallest possible strategy is called the *empty strategy* and written $\perp$. It is formally defined by $\{\epsilon\}$, which corresponds to a strategy that never responds.

Remark 2.52 There is an alternative definition for strategies in which a prefix-closed set is used as opposed to the above definition which relies on *even-length prefix*-closed sets. If σ denotes a strategy in the sense of Def. 2.51 then the corresponding strategy in the alternative definition is given by $\sigma \cup \mathsf{dom}(\sigma)$ where $\mathsf{dom}(\sigma)$ is the domain of σ defined as

$$\mathsf{dom}(\sigma) = \{sa \in P_A^{odd} \mid \exists b.sab \in \sigma\} \ .$$

Copy-cat strategy For every game A there is a strategy id_A on the game $A \multimap A$ called the *copy-cat strategy*. We write A_1 and A_2 to denote the first and second copies of the sub-game A of $A \multimap A$.

Let A be one of the arena A_1 or A_2. We write $A^\perp$ to denote the game A_1 if $A = A_2$ and A_2 otherwise. The copy-cat strategy proceeds as follows: Whenever P has to respond to an O-move played in A, it first replicates this move in the game $A^\perp$. O then responds in $A^\perp$ and finally P replicates O's response back in A.

It is formally defined by:

$$id_A = \{ s \in P^{\mathsf{even}}_{A \multimap A} \mid \forall t \leqslant^{\mathsf{even}} s \,.\, t \upharpoonright A_1 = t \upharpoonright A_2 \} \ ,$$

where P^{even}_A denotes the set of valid positions of even length in the game A, and '$t \leqslant^{\mathsf{even}} s$' denotes that t is an even-length prefix of s.

The copy-cat strategy is also called the ***identity strategy*** on A because it acts as the unit for the operation of strategy composition defined in the next paragraph.

Example 2.53. (a) The copy-cat strategy on $\mathbb{N}$ is given by the following generic play:

$$\begin{array}{ccc}
\mathbb{N} & \multimap & \mathbb{N} \\
 & & q \\
q & & \\
n & & \\
 & & n
\end{array}$$

(This type of diagram was originally introduced to represent plays but as we see here, by giving a generic play, it can also be used to represent a strategy.)

(b) The copy-cat strategy on $\mathbb{N} \multimap \mathbb{N}$ is illustrated by the following diagram:

$$\begin{array}{ccccccc}
(\mathbb{N} & \multimap & \mathbb{N}) & \multimap & (\mathbb{N} & \multimap & \mathbb{N}) \\
 & & & & & & q \\
 & & q & & & & \\
q & & & & & & \\
 & & & & q & & \\
 & & & & m & & \\
m & & & & & & \\
 & & n & & & & \\
 & & & & & & n
\end{array}$$

2.3.2.6 Composition

One of the salient features of game-semantic models is *compositionality*, the ability to compute the denotation of a composite program by composing the denotation of its constituent programs. This notion of composition happens at the level of strategies. We now formally define this operation.

Definition 2.54 (Interaction sequence)**.** Let u be a sequence of moves from games A, B and C together with justification pointers attached to all moves except those that are initial in C. The ***projection*** of s on the game $A \multimap B$, written $u \upharpoonright A, B$ is the subsequence of s obtained by removing from u the moves in C and pointers to moves in C. The projection on $B \multimap C$ is defined similarly.

An ***interaction sequence*** is a sequence of moves with pointers from A, B and C such that $u \upharpoonright A, B$ and $u \upharpoonright B, C$ are legal positions of the game $A \to B$ and $B \to C$ respectively. We write $Int(A, B, C)$ for the set of all such sequences.

We define the projection on the game $A \multimap C$ as follows: $u \upharpoonright A, C$ is the subsequence of u consisting of the moves from A and C with some additional pointers: we add a pointer from $a \in A$ to $c \in C$ whenever a points to some move $b \in B$ itself pointing to c; all the pointers to moves in B are removed.

Given two strategies $\sigma : A \multimap B$ and $\tau : B \multimap C$, the ***interaction*** $\sigma \| \tau$ of σ and τ is defined as the set of interaction sequences that unfold according to the strategy σ in the A, B-projection of the game and to μ in the B, C-projection:

$$\sigma \| \tau = \{ u \in Int(A, B, C) \mid u \upharpoonright A, B \in \sigma \wedge u \upharpoonright B, C \in \tau \} \ .$$

Strategy composition is performed by "parallel composition plus hiding" as defined in the trace semantics of CSP [Hoa83]. Formally,

Definition 2.55 (Strategy composition)**.** Let $\sigma : A \multimap B$ and $\tau : B \multimap C$ be two strategies. The ***composite*** $\sigma; \tau$ is defined as:

$$\sigma; \tau = \{ u \upharpoonright A, C \mid u \in \sigma \| \tau \} \ .$$

It can be verified that composition is well-defined, associative and that the copy-cat strategy id_A is the identity for composition [HO00].

2.3.2.7 Constraint on strategies

Different classes of strategies will be considered depending on the features of the language that we want to model. Here is a list of restrictions that are commonly considered:

- *Well-bracketing:* We call *pending question* the last question in a sequence that has not been answered. A strategy σ is well-bracketed if for every play $s \cdot m \in \sigma$ where m is an answer, m points to the pending question in s.

- *History-free strategies:* A strategy is history-free if the Proponent's move at any position of the game where he has to play is determined by the last move of the Opponent (*i.e.*, P ignores the complete history up the last move).

- *History-sensitive strategies:* The Proponent follows a history-sensitive strategy if he needs to have access to the full history of the moves in order to decide which move to make.

- *Innocence:* In these strategies, the Proponent determines his next move based solely on a restricted view of the history of the play, namely the P-view at that point. It always plays the same move with the same pointer for a given P-view. Innocence plays an important role in the modeling of purely functional languages.

The formal definition of innocence is:

Definition 2.56 (Innocence)**.** Given positions $sab, ta \in L_A$ where sab has even length and $\ulcorner sa \urcorner = \ulcorner ta \urcorner$, there is a unique extension of ta by the move b together with a justification pointer such that $\ulcorner sab \urcorner = \ulcorner tab \urcorner$. We write this extension $\mathsf{match}(sab, ta)$.

The strategy $\sigma : A$ is ***innocent*** if and only if:

$$\begin{pmatrix} \ulcorner sa \urcorner = \ulcorner ta \urcorner \\ sab \in \sigma \\ t \in \sigma \wedge ta \in P_A \end{pmatrix} \quad \Longrightarrow \quad \mathsf{match}(sab, ta) \in \sigma \ .$$

Since the next move is determined by the P-view, an innocent strategy induces a partial function mapping P-views to P-moves called the ***view function***. Not every partial function from P-views to P-moves gives rise to an innocent strategy, however. (Hyland and Ong gave a sufficient condition [HO00].)

2.3.3 Categorical interpretation

This section recalls briefly the categorical interpretation of games [McC96a, HO00, AMJ94]. We consider the category [Cro93] $\mathcal{G}$ whose objects are games and morphisms are strategies. A morphism from A to B is a strategy on the game $A \multimap B$. Composition of morphisms is given by strategy composition. We also consider sub-categories of $\mathcal{G}$ corresponding to various restrictions imposed on strategies: $\mathcal{G}_i$ is the sub-category whose morphisms are the innocent strategies, $\mathcal{G}_b$ has only the well-bracketed strategies and $\mathcal{G}_{ib}$ has the innocent and well-bracketed strategies.

Proposition 2.57. *$\mathcal{G}$, $\mathcal{G}_i$, $\mathcal{G}_b$ and $\mathcal{G}_{ib}$ are categories.*

In particular this means that composition of strategies is well-defined, associative, has a unit (the copy-cat strategy), preserves innocence and well-bracketedness [HO00, AMJ94].

2.3.3.1 Monoidal structure

In Sec. 2.3.2.3 we have defined the tensor product on games. We now define the corresponding transformation on morphisms. Given two strategies $\sigma : A \multimap B$ and $\tau : C \multimap D$ the strategy $\sigma \otimes \tau : (A \otimes C) \multimap (B \otimes D)$ is defined by:

$$\sigma \otimes \tau = \{ s \in L_{A \otimes C \multimap B \otimes D} \; s \upharpoonright A, B \in \sigma \wedge s \upharpoonright C, D \in \tau \} \ .$$

It can be shown that the tensor product is associative, commutative and has $I = \langle \emptyset, \emptyset, \emptyset, \{\epsilon\} \rangle$ as identity. Hence the game category $\mathcal{G}$ is a symmetric monoidal category. Moreover $\mathcal{G}_i$ and $\mathcal{G}_b$ are sub-symmetric monoidal categories of $\mathcal{G}$, and $\mathcal{G}_{ib}$ is a sub-symmetric monoidal category of $\mathcal{G}_i$, $\mathcal{G}_b$ and $\mathcal{G}$.

2.3.3.2 Closed structure

Let A, B and C be three games. Given a strategy on $A \otimes B \multimap C$ we can clearly convert it into a strategy on $A \multimap (B \multimap C)$ by performing the appropriate retagging of the moves. This transformation defines an isomorphism written Λ_B and called ***currying***. Thus the hom-set $\mathcal{G}(A \otimes B, C)$ is isomorphic to the hom-set $\mathcal{G}(A, B \multimap C)$, which makes $\mathcal{G}$ an autonomous (*i.e.*, symmetric monoidal closed) category. The categories $\mathcal{G}_i$ and $\mathcal{G}_b$ are sub-autonomous categories of $\mathcal{G}$, and $\mathcal{G}_{ib}$ is a sub-autonomous category of $\mathcal{G}_i$, $\mathcal{G}_b$ and $\mathcal{G}$.

We write $ev_{A,B} : (A \multimap B) \otimes A \rightarrow B$ to denote the ***evaluation strategy*** obtained by uncurrying the identity map on $A \rightarrow B$. The evaluation strategy is in fact the copy-cat strategy for the game $(A \multimap B) \otimes A \rightarrow B$.

2.3.3.3 Cartesian product

The cartesian product from Sec. 2.3.2.3 defines indeed a cartesian product in the category $\mathcal{G}$, $\mathcal{G}_i$, $\mathcal{G}_b$ and $\mathcal{G}_{ib}$. The projections $\pi_1 : A \times B \rightarrow A$ and $\pi_1 : A \times B \rightarrow B$ are given by the obvious copy-cat strategies. Given two category morphisms $\sigma : C \rightarrow A$ and $\tau : C \rightarrow B$, the ***pairing*** morphism $\langle \sigma, \tau \rangle : C \rightarrow A \times B$ is given by:

$$\begin{aligned} \langle \sigma, \tau \rangle &= \{ s \in L_{C \multimap A \times B} \mid s \upharpoonright C, A \in \sigma \wedge s \upharpoonright B = \epsilon \} \\ &\cup \ \{ s \in L_{C \multimap A \times B} \mid s \upharpoonright C, B \in \tau \wedge s \upharpoonright A = \epsilon \} \ . \end{aligned}$$

2.3.3.4 Cartesian closed structure

To obtain a cartesian closed category it remains to define a *terminal* object as well as the *exponential* construct for every two games A and B. The category $\mathcal{G}$ itself is not cartesian closed but it is possible to define a new category of games that is cartesian closed.

For every game A the ***exponential*** game $!A$ is given by:

$$\begin{aligned} M_{!A} &= M_A \\ \lambda_{!A} &= \lambda_A \\ \vdash_{!A} &= \vdash_A \\ P_{!A} &= \{s \in L_{!A} | \text{ for each initial move } m,\ s \upharpoonright m \in P_A\} \ . \end{aligned}$$

Think of it as the multi-threaded version of the game A in which a new copy the game can be spawned at any time. Plays of $!A$ are thus interleavings of plays of A. We have the following identities:

$$\begin{aligned} !(A \times B) &= !A \otimes !B \\ \mathbf{1} &= !\mathbf{1} \ . \end{aligned}$$

A game A is said to be ***well-opened*** if for every position $s \in P_A$ the only initial move in s is the first one. In a well-opened game, plays contain a single "thread" of moves. Given a strategy on a well-opened game, one can turn it into a "multi-threaded" strategy using the *promotion* operator:

Definition 2.58 (Promotion)**.** Consider a well-opened game B. Given a strategy on $!A \multimap B$, its ***promotion*** $\sigma^\dagger : !A \multimap !B$ is the strategy which plays several copies of σ. Formally:

$$\sigma^\dagger = \{s \in L_{!A \multimap !B} \mid \text{ for all initial } m,\ s \upharpoonright m \in \sigma\} \ .$$

It can be shown that promotion is a well-defined strategy and that it preserves innocence and well-bracketing. We now introduce the category of well-opened games:

Definition 2.59 (Category of well-opened games)**.** The category $\mathcal{C}$ of well-opened games, also called the *co-Kleisli category* of $\mathcal{G}$, is defined as follows:

- The objects are the well-opened games.
- A morphism $\sigma : A \to B$ is a strategy for the game $!A \multimap B$,
- The identity map for A is the copy-cat strategy on $!A \multimap A$ (which is well-defined for well-opened games). It is called dereliction, denoted by der_A and defined formally by:

$$\mathsf{der}_A = \{s \in P^{\mathsf{even}}_{!A \multimap A} \mid \forall t \leqslant^{\mathsf{even}} s \ . \ t \upharpoonright !A = t \upharpoonright A\} \ .$$

- Composition of morphisms $\sigma : !A \multimap B$ and $\tau : !B \multimap C$ denoted by $\sigma \fatsemi \tau : !A \multimap C$ is defined as $\sigma^\dagger ; \tau$.

$\mathcal{C}$ is a well-defined category and has three sub-categories $\mathcal{C}_i$, $\mathcal{C}_b$, $\mathcal{C}_{ib}$ corresponding respectively to sub-category of innocent, well-bracketed, and innocent well-bracketed strategies.

The empty game $\mathbf{1}$ is a terminal object for the category $\mathcal{C}$. Further for every two games A and B, we define their product as $A \times B$ and their exponential as $!A \multimap B$. The hom-sets $\mathcal{C}(A \times B, C)$ and $\mathcal{C}(A, !B \multimap C)$ are isomorphic. Indeed:

$$\begin{aligned} \mathcal{C}(A \times B, C) &= \mathcal{G}(!(A \times B), C) \\ &= \mathcal{G}(!A \otimes !B, C) \\ &\cong \mathcal{G}(!A, !B \multimap C) \qquad (\mathcal{G} \text{ is a closed monoidal category}) \\ &= \mathcal{C}(A, !B \multimap C) \ . \end{aligned}$$

Hence $\mathcal{C}$ is a cartesian closed category. Furthermore $\mathcal{C}_i$ and $\mathcal{C}_b$ are sub-cartesian closed categories of $\mathcal{C}$, and $\mathcal{C}_{ib}$ is as sub-cartesian closed category of each of $\mathcal{C}$, $\mathcal{C}_i$ and $\mathcal{C}_b$.

2.3.3.5 Order enrichment

Strategies can be ordered using the inclusion ordering. Under this ordering, the set of strategies on a given game A is a pointed directed complete partial order; the least upper bound is given by the set-theoretic union and the least element is the empty strategy $\{\epsilon\}$.

Moreover all the operators on strategies that we have defined so far (composition, tensor product, etc.) are continuous. Hence the categories $\mathcal{C}$ and $\mathcal{G}$ are cpo-enriched.

2.3.3.6 Intrinsic preorder

Let Σ denote the *Sierpinski game* with a single question q and single answer a. There are only two strategies on Σ: $\perp = \{\epsilon\}$ and $\top = \{\epsilon, qa\}$, both innocent and well-bracketed. For every object A, the ***intrinsic preorder*** $\lesssim_A$ on the set of strategies on the game A is defined by:

$$\sigma \lesssim_A \tau \quad \iff \quad \forall \alpha : A \rightarrow \Sigma.\ \sigma \fatsemi \tau = \top \implies \tau \fatsemi \alpha = \top\ .$$

This indeed defines a preorder [AMJ94]. The ***quotiented category*** $\mathcal{C}/\lesssim$ is defined as follows. The objects of $\mathcal{C}/\lesssim$ are those of $\mathcal{C}$, and the morphisms are the equivalence classes of morphisms in $\mathcal{C}$ modulo the equivalence relation induced by $\lesssim$.

We will consider the quotiented categories $\mathcal{C}_\$/\lesssim_\$$ where \$ ranges in $\{i, b, ib\}$. (The full abstraction of the game-semantic model of PCF holds in the quotiented category $\mathcal{C}_{ib}/\lesssim_{ib}$ rather than $\mathcal{C}_{ib}$.)

2.3.4 The fully abstract game model of PCF

In this section we show how game semantics can be used to model the programming language PCF and we recall the full abstraction result [HO00].

It is well known that cartesian closed categories are models of typed lambda calculi. We have just seen in the previous section that games and strategies form a cartesian closed category, they can therefore be used to model typed lambda-calculi.

The idea is as follows. The game played is induced by the type of the term. The Opponent (O) incarnates the environment while the Proponent (P) incarnates the term to model. The Proponent's strategy is determined by the term itself; it is computed inductively on its syntax. This means that O is responsible of providing the values of the term's input parameters, whereas P is responsible for performing the computation of the term itself. A play of the game unfolds as follows: The Opponent opens the game by asking the question "What is the result of the execution of the term?". The Proponent may then request further information by asking questions such as "What is the input given to the term?"; O can provide P with an answer—the value of the input—or can continue by asking another question. This dialog goes on until O obtains an answer to his initial question.

2.3.4.1 Modeling the simple types

Each simple type A is interpreted by a game from the category $\mathcal{C}$ denoted $[\![A]\!]$. A program context $\Gamma = x_1 : A_1, \ldots x_n : A_n$ is interpreted by the game $[\![\Gamma]\!] = [\![A_1]\!] \times \ldots \times [\![A_n]\!]$. The empty context is interpreted by the terminal object $\mathbf{1}$ of the cartesian closed category $\mathcal{C}$: $[\![\emptyset]\!] = \mathbf{1}$.

The base type `exp` is interpreted by the flat game $\mathbb{N}$ over the natural number. Given the interpretation of the base type, the interpretation of the function space type $A \rightarrow B$ is given by the exponential object of $[\![A]\!]$ and $[\![B]\!]$ in the cartesian closed category $\mathcal{C}$:

$$[\![A \rightarrow B]\!] = ![\![A]\!] \multimap [\![B]\!]\ .$$

2.3.4.2 Lambda calculus fragment

A term-in-context $\Gamma \vdash M : A$ is interpreted in the model by a strategy on the game $[\![\Gamma]\!] \to [\![A]\!]$.

For instance take the game $[\![\mathtt{exp}]\!]$. It has only one question (the initial O-question) and P-moves are answers corresponding to the natural numbers. There exist only two kinds of strategies for the game $[\![\mathtt{exp}]\!]$:

(i) The empty strategy where P never answer the initial question. This corresponds to a non-terminating computation;

(ii) The strategies where P always answers by playing the same number n. This models a numerical constant of the language.

The strategy denotation of a term-in-context is defined inductively on the structure of the term:

- Variables are interpreted by projection:

$$[\![x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i]\!] = \pi_i : [\![A_i]\!] \times \ldots \times [\![A_i]\!] \times \ldots \times [\![A_n]\!] \to [\![A_i]\!] \ .$$

- Abstraction: The term-in-context $\Gamma \vdash \lambda x^A.M : A \to B$ is modeled by a morphism $[\![\Gamma]\!] \to (![\![A]\!] \multimap [\![B]\!])$ obtained by currying:

$$[\![\Gamma \vdash \lambda x^A.M : A \to B]\!] = \Lambda([\![\Gamma, x : A \vdash M : B]\!]) \ .$$

- Application is modeled using the evaluation map $ev_{A,B} : (!A \multimap B) \times A \to B$:

$$[\![\Gamma \vdash MN : B]\!] = \langle [\![\Gamma \vdash M : A \to B, \Gamma \vdash N : A]\!] \rangle \fatsemi ev_{A,B} \ .$$

Example 2.60 (Kierstead terms)**.** In Sec. 2.3.6 we have shown that there exist two different strategies on the game $[\![((\mathbb{N}^1 \to \mathbb{N}^2) \to \mathbb{N}^3) \to \mathbb{N}^4]\!]$ containing a play whose underlying sequence of move is $q^4 q^3 q^2 q^3 q^2 q^1$ but whose justification pointers differ.

These two strategies are precisely the denotation of the *Kierstead terms* defined as follows:

$$\begin{aligned} M_1 &\equiv \lambda f.f(\lambda x.f(\lambda y.y)) : ((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}) \to \mathbb{N} \\ M_2 &\equiv \lambda f.f(\lambda x.f(\lambda y.x)) : ((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}) \to \mathbb{N} \ . \end{aligned}$$

Suppose that q^1 is justified by the first occurrence of q^2 then it means that the Proponent is requesting the value of the variable x bound in the subterm $\lambda x.f(\lambda y....)$. If P needs to know the value of x, this means that P follows the strategy induced by the subterm $\lambda y.x$: this corresponds to a play of the strategy $[\![M_2]\!]$. Otherwise q^1 is justified by the second occurrence of q^2, which corresponds to a play of $[\![M_1]\!]$.

2.3.4.3 PCF fragment

We now show how to model PCF constructs. In the following, we tag the sub-arenas of the games considered to make it possible to distinguish identical arenas from different components of the game. We also tag moves (in exponent) to identify the component in which the move belongs. We will omit the pointers in the play when no ambiguity arise.

The arithmetic constants of PCF are interpreted as follows:

- The successor arithmetic operator is modeled by the following strategy on $[\![\mathbb{N}^1 \to \mathbb{N}^0]\!]$:

$$[\![\mathtt{succ}]\!] = \mathsf{Pref}^{\mathsf{even}}\{q^0 \cdot q^1 \cdot n^1 \cdot (n+1)^0 \mid n \in \mathbb{N}\} \ .$$

  where $\mathsf{Pref}^{\mathsf{even}} X$ denotes the set consisting of the prefixes of even length of plays of X.

- The predecessor arithmetic operator is denoted by the strategy

$$[\![\mathtt{pred}]\!] = \mathsf{Pref}^{\mathsf{even}}\left(\{q^0 \cdot q^1 \cdot n^1 \cdot (n-1)^0 \mid n > 0\} \cup \{q^0 \cdot q^1 \cdot 0^1 \cdot 0^0\}\right) \ .$$

- Given a term-in-context $\Gamma \vdash \texttt{succ}\ M : \texttt{exp}$ we define:

$$[\![\Gamma \vdash \texttt{succ}\ M : \texttt{exp}]\!] = [\![\Gamma \vdash M : \texttt{exp}]\!] \fatsemi [\![\texttt{succ}]\!]$$

$$[\![\Gamma \vdash \texttt{pred}\ M : \texttt{exp}]\!] = [\![\Gamma \vdash M : \texttt{exp}]\!] \fatsemi [\![\texttt{pred}]\!] \ .$$

- The conditional operator is denoted by the following strategy on $[\![\mathbb{N}^3 \times \mathbb{N}^2 \times \mathbb{N}^1 \to \mathbb{N}^0]\!]$:

$$[\![\texttt{cond}]\!] = \mathsf{Pref}^{\mathsf{even}}\{q^0 \cdot q^3 \cdot 0 \cdot q^2 \cdot n^2 \cdot n^0 \mid n \in \mathbb{N}\} \cup \mathsf{Pref}^{\mathsf{even}}\{q^0 \cdot q^3 \cdot m \cdot q^2 \cdot n^2 \cdot n^0 \mid m > 0, n \in \mathbb{N}\} \ .$$

  Given a term-in-context $\Gamma \vdash \texttt{cond}\ M\ N_1\ N_2 : \texttt{exp}$ we define:

$$[\![\Gamma \vdash \texttt{cond}\ M\ N_1\ N_2 : \texttt{exp}]\!] = \langle [\![\Gamma \vdash M : \texttt{exp}]\!], [\![\Gamma \vdash N_1 : \texttt{exp}]\!], [\![\Gamma \vdash N_2 : \texttt{exp}]\!] \rangle \fatsemi [\![\texttt{cond}]\!] \ .$$

The interpretation of the Y combinator is slightly more complicated. Consider the term $\Gamma \vdash M : A \to A$. Its denotation f is a morphism $[\![\Gamma]\!] \times [\![A]\!] \to [\![A]\!]$. We define the chain g_n of morphisms $[\![\Gamma]\!] \to [\![A]\!]$ as follows:

$$\begin{aligned} g_0 &= \perp \\ g_{n+1} &= F(g_n) = \langle id_{[\![\Gamma]\!]}, g_n \rangle \fatsemi f \end{aligned}$$

where $\perp$ denotes the empty strategy $\{\epsilon\}$. It is easy to see that $(g_n)_{n \in \mathbb{N}}$ forms a chain. The denotation $[\![\mathsf{Y}M]\!]$ is defined as the least upper bound of the chain g_n which is also the least fixed point of F. Its existence is guaranteed by the fact that the category of games is cpo-enriched.

Since all the strategies encountered up to now are innocent and well-bracketed, the game model of PCF can be interpreted in any of the four categories $\mathcal{C}$, $\mathcal{C}_i$, $\mathcal{C}_b$, $\mathcal{C}_{ib}$. The category $\mathcal{C}_{ib}$ is referred as the ***intensional game model*** of PCF.

2.3.4.4 Observational preorder

A context denoted $C[-]$ is a term containing a hole denoted by the symbol '$-$'. If $C[-]$ is a context then $C[M]$ denotes the term obtained after replacing the hole by the term M. $C[M]$ is well-formed provided that M has the appropriate type. This substitution is done capture-permitting, as opposed to the capture-avoiding substitution used to contract beta-redexes in the lambda calculus.

Definition 2.61. The ***observational preorder*** is a relation $\sqsubseteq$ on terms defined as follows: For every two closed terms M and N of the same type,

$$M \sqsubseteq N \iff \text{for all context } C[-] \text{ such that } C[M] \text{ and } C[N] \text{ are well-formed closed PCF term of type } \texttt{exp},\ C[M] \Downarrow \text{ implies } C[N] \Downarrow \ .$$

The reflexive closure of $\sqsubseteq$, denoted $\cong$, is called the ***observational equivalence*** relation.

The intuition behind this definition is that two terms are observationally equivalent if there is no context that distinguishes them; in which case they can be safely interchanged in any program context.

2.3.4.5 Soundness

We say that a model is *sound for evaluation* if the denotation of a term is preserved by the evaluation relation $\Downarrow$ of the big-step semantics of the language. For every term M and value V we have:

$$M \Downarrow V \quad \Longrightarrow \quad [\![M]\!] = [\![V]\!] \ .$$

Lemma 2.62 ([AM98b])**.** *The game model of PCF is sound for evaluation.*

Definition 2.63 (Computable terms)**.**

- A closed term $\vdash M : B$ of base type is computable if $[\![M]\!] \neq \bot$ implies $M \Downarrow$.
- A higher-order closed term $\vdash M : A \to B$ is computable if MN is computable for every computable closed term $\vdash N : A$.
- An open term $x_1 : A_1, \ldots, x_n : A_n \vdash M : A \to B$ is computable if $\vdash M[N_1/x_1, \ldots N_n/x_n]$ is computable for all computable closed terms $N_1 : A_1, \ldots, N_n : A_n$.

A model is ***computationally adequate*** if all terms are computable.

Lemma 2.64 ([AM98b])**.** *The game model of PCF is computationally adequate.*

A model of a programming language is said to be ***sound*** if whenever the denotation of two programs are equal then the two programs are observationally equivalent; formally for every closed terms M and N of the same type we have:

$$[\![M]\!] = [\![N]\!] \implies M \cong N \ .$$

Soundness is the least condition one can require from a model of programming language: it guarantees that we can reason about terms by manipulating objects in the denotational model.

The model is said to be ***inequationally sound*** if the following stronger condition holds

$$[\![M]\!] \subseteq [\![N]\!] \implies M \lesssim N \ .$$

The inequational soundness of the game model of PCF follows from the last two lemmas:

Proposition 2.65. *The game model of PCF is* inequationally sound.

Proof. Take two closed PCF terms M and N. Suppose that $[\![M]\!] \subseteq [\![N]\!]$ then by compositionality of the model we have $[\![C[M]]\!] \subseteq [\![C[N]]\!]$. Suppose that $C[M] \Downarrow$ for some context $C[-]$ then by soundness (Lemma 2.62) we have $[\![C[M]]\!] \neq \bot$, which implies $[\![C[N]]\!] \neq \bot$. The adequacy of the model (Lemma 2.64) then gives us $C[N] \Downarrow$. Hence $M \lesssim N$. $\square$

2.3.4.6 Definability

We now work in the category $\mathcal{C}_{ib}$ of innocent and well-bracketed strategies. The *definability* property is the key to the full-abstraction result. It says that every *compact* element of the model is the denotation of some term. In $\mathcal{C}_{ib}$, the ***compact morphisms*** are the innocent strategies with finite view-function. Due to its economical syntax, PCF does not satisfy the definability result: there are strategies that are not the denotation of any term in PCF. For instance consider the *ternary conditional* strategy acting as follows: It tests the value of its first parameter, if it is equal to 0 or 1 then it returns the value of the second or third parameter respectively, otherwise it returns the value of the fourth parameter. This is illustrated in the left diagram of Fig. 2.3.4.6. Such computation can be *operationally* simulated in PCF by the term $T_3 = \texttt{cond}\ M\ N_1(\texttt{cond}\ (\texttt{pred}\ M)\ N_2\ N_3)$. The term T_3, however, is not denoted by the ternary conditional strategy. Its denotation is instead given by the right diagram on Fig. 2.3.4.6.

In PCF$_c$, however, the ternary conditional strategy is definable by the term *case*$_3$. In fact, the definability result holds for PCF$_c$:

Proposition 2.66 (Definability)**.** *Let A be a PCF type and σ be a compact innocent and well-bracketed strategy on A. There exists a PCF$_c$ term M such that $[\![M]\!] = \sigma$.*

The definability only holds for PCF$_c$ but this suffices to prove full abstraction of PCF. This is because the $\texttt{case}_k$ constructs of PCF$_c$ can all be simulated by PCF terms with the same operational semantics, and consequently PCF$_c$ is a conservative extension of PCF (*i.e.*, if M and N are terms such that for every PCF-context $C[-]$, $C[M] \Downarrow \implies C[N] \Downarrow$ then the same is true for every PCF$_c$-context.)

$$
\begin{array}{ccccccccc}
!\mathbf{N} & \otimes & !\mathbf{N} & \otimes & !\mathbf{N} & \otimes & !\mathbf{N} & \multimap & !\mathbf{N} \\
 & & & & & & & & q \\
q & & & & & & & & \\
0 & & & & & & & & \\
 & & q & & & & & & \\
 & & n & & & & & & \\
 & & & & & & & & n \\
\hline
 & & & & & & & & q \\
q & & & & & & & & \\
1 & & & & & & & & \\
 & & & & q & & & & \\
 & & & & n & & & & \\
 & & & & & & & & n \\
\hline
 & & & & & & & & q \\
q & & & & & & & & \\
m > 1 & & & & & & & & \\
 & & & & & & q & & \\
 & & & & & & n & & \\
 & & & & & & & & n
\end{array}
\qquad
\begin{array}{ccccccccc}
!\mathbf{N} & \otimes & !\mathbf{N} & \otimes & !\mathbf{N} & \otimes & !\mathbf{N} & \multimap & !\mathbf{N} \\
 & & & & & & & & q \\
q & & & & & & & & \\
0 & & & & & & & & \\
 & & q & & & & & & \\
 & & n & & & & & & \\
 & & & & & & & & n \\
\hline
 & & & & & & & & q \\
q & & & & & & & & \\
1 & & & & & & & & \\
q & & & & & & & & \\
0 & & & & & & & & \\
 & & & & q & & & & \\
 & & & & n & & & & \\
 & & & & & & & & n \\
\hline
 & & & & & & & & q \\
q & & & & & & & & \\
m > 1 & & & & & & & & \\
q & & & & & & & & \\
m - 1 > 0 & & & & & & & & \\
 & & & & & & q & & \\
 & & & & & & n & & \\
 & & & & & & & & n
\end{array}
$$

Figure 2.1: Strategy denotation of $case_3$ (left) and T_3 (right).

2.3.4.7 Full abstraction

The converse of soundness is called *completeness*. A model is ***complete*** if:

$$M \cong N \implies [\![M]\!] = [\![N]\!] \ .$$

Further, if the stronger relation

$$M \sqsubseteq\!\!\!\!\sim N \implies [\![M]\!] \subseteq [\![N]\!]$$

holds then the model is said to be ***inequationally complete***.

A model is ***fully abstract*** if it is both sound and complete, and ***inequationally fully abstract*** if it is inequationally sound and inequationally complete.

Full abstraction of PCF cannot be stated directly in the category $\mathcal{C}_{ib}$. Instead we need to consider the quotiented category $\mathcal{C}_{ib}/\lesssim_{ib}$. But first we need to make sure that $\mathcal{C}_{ib}/\lesssim_{ib}$ is a model of PCF. $\mathcal{C}_{ib}/\lesssim_{ib}$ is a poset-enriched cartesian closed category. The denotation of the basic types and constants of PCF can be transposed from $\mathcal{C}_{ib}$ to $\mathcal{C}_{ib}/\lesssim_{ib}$. Although it is not known whether $\mathcal{C}_{ib}/\lesssim_{ib}$ is enriched over the category of CPOs, it can be proved that it satisfies a condition called *rationality* [AMJ94] and this suffices to ensure that $\mathcal{C}_{ib}/\lesssim_{ib}$ is indeed a model of PCF. This category will be referred as the ***extensional game model*** of PCF. The full abstraction of the game model then follows from Proposition 2.65 and 2.66:

Theorem 2.67 (Full abstraction [AMJ94, HO00, Nic94]). *Let M and N be two closed PCF terms. Then*

$$[\![M]\!] \lesssim_{ib} [\![N]\!] \iff M \sqsubseteq\!\!\!\!\sim N \ ,$$

where $\lesssim_{ib}$ denotes the intrinsic preorder of the category $\mathcal{C}_{ib}$.

2.3.5 The fully abstract game model of Idealized Algol

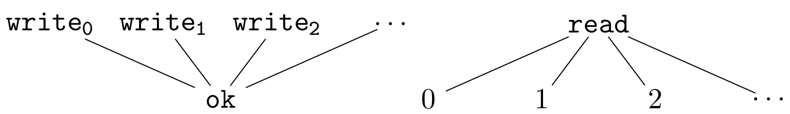
We now describe the fully abstract game model of IA [AM99].

All the strategies used to model PCF are well-bracketed and innocent. To obtain a model of IA, however we need to introduce strategies that are not innocent. This is necessary to model the memory cell variable created with the `new` operator. The intuition is that a cell needs to remember the last value which was written in it in order to be able to return it when it is subsequently read, and this can only be done by looking at the whole history of moves, not only those present in the P-view. We therefore restrict our attention to the categories $\mathcal{C}$ and $\mathcal{C}_b$.

Base types

The type `com` is modeled by the flat game with a single initial question `run` and a single answer `done`. The idea is that O can request the execution of a command by playing `run`, P then executes the command and if it terminates, acknowledges it by playing `done`.

The variable type `var` is modeled by the game $\texttt{com}^\omega \times \texttt{exp}$ illustrated below:



Modelling the constants

- The constant `skip` is interpreted by the strategy $\{\epsilon, \texttt{run} \cdot \texttt{done}\}$.

- Sequential composition $\mathtt{seq_{exp}}$ is interpreted by the following strategy:

$$
\begin{array}{ccccc}
!\mathtt{com} & \otimes & !\mathtt{exp} & \stackrel{\mathtt{seq_{exp}}}{\multimap} & \mathtt{exp} \\
 & & & & q \\
\mathtt{run} & & & & \\
\mathtt{done} & & & & \\
 & & q & & \\
 & & n & & \\
 & & & & n
\end{array}
$$

- Assignment $\mathtt{assign}$ and dereferencing $\mathtt{deref}$ are denoted by the following strategies:

$$
\begin{array}{ccccc}
!\mathtt{var} & \otimes & !\mathtt{exp} & \stackrel{\mathtt{assign}}{\multimap} & \mathtt{com} \\
 & & & & q \\
 & & q & & \\
 & & n & & \\
\mathtt{write}_n & & & & \\
\mathtt{ok} & & & & \\
 & & & & \mathtt{done}
\end{array}
\qquad
\begin{array}{ccc}
!\mathtt{var} & \stackrel{\mathtt{deref}}{\multimap} & \mathtt{exp} \\
 & & q \\
\mathtt{read} & & \\
n & & \\
 & & n
\end{array}
$$

- $\mathtt{mkvar}$ is modeled by the paired strategy $\langle \mathtt{mkvar}_{acc}, \mathtt{mkvar}_{exp} \rangle$ where $\mathtt{mkvar}_{acc}$ and $\mathtt{mkvar}_{exp}$ are the following strategies:

$$
\begin{array}{ccccccc}
!(!\mathtt{exp} & \multimap & \mathtt{com}) & \otimes & !\mathtt{exp} & \stackrel{\mathtt{mkvar}_{acc}}{\multimap} & \mathtt{com}^{\omega} \\
 & & & & & & \mathtt{write}_n \\
 & & \mathtt{run} & & & & \\
q & & & & & & \\
n & & & & & & \\
 & & \mathtt{done} & & & & \\
 & & & & & & \mathtt{ok}
\end{array}
\qquad
\begin{array}{ccccccc}
!(!\mathtt{exp} & \multimap & \mathtt{com}) & \otimes & !\mathtt{exp} & \stackrel{\mathtt{mkvar}_{exp}}{\multimap} & \mathtt{exp} \\
 & & & & & & \mathtt{read} \\
 & & & & q & & \\
 & & & & n & & \\
 & & & & & & n
\end{array}
$$

- Block-allocated variable ($\mathtt{new}$): The strategies introduced until now are all innocent. In order to model the $\mathtt{new}$ operator, it is necessary to introduce non-innocent strategies, also called *knowing strategies*. We call ***memory-cell strategy*** the knowing well-bracketed strategy written $cell : I \multimap !\mathtt{var}$ behaving as follows: It responds to $\mathtt{write}$ with $\mathtt{ok}$ and to $\mathtt{read}$ with the last value written or 0 if no value has been written yet. The denotation of a term-in-context $\Gamma \vdash \mathtt{new}\ x\ \mathtt{in}\ M : A$ is then defined as the strategy:

$$
[\![\Gamma \vdash \mathtt{new}\ x\ \mathtt{in}\ M : A]\!] = (id_{[\![\Gamma]\!]} \otimes cell) \fatsemi [\![\Gamma, x : \mathtt{var} \vdash M : A]\!] : !\Gamma \multimap \mathtt{com}\ .
$$

Full abstraction

Inequational soundness can also be shown for IA. Proving soundness of the evaluation requires slightly more work than in the PCF case due to the fact that stores need to be made explicit. Also, one needs to define an appropriate notion of *computable term* that takes into account the presence of stores in the evaluation semantics. It is also possible to prove that the model is computational adequate. We then have:

Proposition 2.68 (Abramksy and McCusker [AMJ94])**.** *The game model of IA is inequationally sound.*

A result called the Innocent Factorization Theorem [AM97] shows that the strategies in $\mathcal{G}_b$ can all be obtained by composing the non-innocent strategy *cell* with some innocent strategy. The strategy *cell* can therefore be viewed as a generic non-innocent strategy. Using this factorization argument, it is possible to prove the definability result:

Proposition 2.69 (Definability). *For every compact well-bracketed strategy σ on a game A denoting an IA type, there exists an IA-term M such that $[\![M]\!] = \sigma$.*

Full abstraction for the model $\mathcal{C}_b$ is then a consequence of inequational soundness and definability:

Theorem 2.70 (Full abstraction). *Let M and N be two closed IA-terms. Then*

$$[\![M]\!] \lesssim_b [\![N]\!] \iff M \sqsubseteq\!\!\!\sim N \ ,$$

where $\lesssim_b$ denotes the intrinsic preorder of the category $\mathcal{C}_b$.

2.3.6 On the necessity of justification pointers

For every legal justified sequence of moves s, we write $?(s)$ to denote the subsequence consisting of the unanswered question moves of s. It is easy to check that if s satisfies alternation then so does $?(s)$.

Lemma 2.71. *If $s \cdot q$ is a legal position (i.e., a justified sequence satisfying visibility and alternation) satisfying well-bracketing and q is a non-initial question then q points in $?(s)$.*

Proof. By induction on the length of $s \cdot q$. The base case $s = \epsilon$ is trivial. Let $s = s \cdot q$, where q is not initial.

Suppose q is a P-move. We prove that q cannot point to an O-question that has been answered. Suppose that an O-move q' occurs before q and is answered by the move a also occurring before q. Then we have $s = s_1 \cdot q'^O \cdot s_2 \cdot a^P \cdot s_3 \cdot q^P$ where a is justified by q'. a is not in the P-view $\ulcorner s_{<q} \urcorner$. Indeed this would imply that some O-move occurring in s_3 points to a, but this is impossible since answer moves are not enablers. Hence the move a must be situated underneath an O-to-P link. Let m denote the link's origin, the P-view of s has the following form: $\ulcorner s \urcorner = \ulcorner s_1 \cdot q'^O \cdot s_2 \cdot a^P \ldots m^O \urcorner \ldots q^P$ where m is an O-move pointing before a.

If m is an answer move then it must point to the last unanswered move (the last move in $?(s_{<m})$). If m is a question move then it is not initial since there is a link going from m. Therefore by the induction hypothesis, m must point to a move in $?(s_{<m})$.

Since s is well bracketed, all the questions in the segment $q' \ldots a$ are answered. Therefore since m points to an unanswered question occurring before a, m must point to a move occurring strictly before q'. Consequently q' does not occur in the P-view $\ulcorner s \urcorner$. By visibility, q must point in the P-view $\ulcorner s \urcorner$ therefore q does not point to q'.

A similar argument holds if q is an O-move. ☐

This means that in a well-bracketed legal position $s \cdot m$ where m is not initial, m's justifier is a question occurring in $?(s)$. Also if m is an answer then its justifier is precisely the *last* question in $?(s)$. Furthermore, if m is a P-move then by visibility it should point to an unanswered question in $\ulcorner m \urcorner$ therefore it should also point in $?(\ulcorner m \urcorner)$. Similarly, if m is a non initial O-move then it points in $?(\llcorner m \lrcorner)$.

Lemma 2.72. *Let s be a legal well-bracketed position.*

(i) If $s = \epsilon$ or if the last move in s is not a P-answer then $?(\ulcorner s \urcorner) = \ulcorner ?(s) \urcorner$;

(ii) If $s = \epsilon$ or if the last move in s is not an O-answer then $?(\llcorner s \lrcorner) = \llcorner ?(s) \lrcorner$.

Proof. (i) By induction on the length of s. The base case is trivial. Step case: Suppose that $s \cdot m$ is a legal well-bracketed position.

- If m is an initial O-question then $?(\ulcorner s \cdot m \urcorner) = ?(m) = m = \ulcorner ?(s) \cdot m \urcorner = \ulcorner ?(s \cdot m) \urcorner$.

- If m is a non initial O-question then $s \cdot m^O = s' \cdot q^P \cdot s'' \cdot m^O$ where m is justified by q. We have $?(\ulcorner s \urcorner) = ?(\ulcorner s' \urcorner \cdot q \cdot m) = ?(\ulcorner s' \urcorner) \cdot q \cdot m$. If s' is not empty then its last move must be an O-move (by alternation), therefore by the induction hypothesis $?(\ulcorner s' \urcorner) = ?(\ulcorner ?(s') \urcorner)$. By the previous lemma, m's justified occurs in $?(s)$ therefore $?(s \cdot m) = ?(s') \cdot q^P \cdot u \cdot m^O$ for some sequence u and thus $\ulcorner ?(s \cdot m) \urcorner = \ulcorner ?(s') \urcorner \cdot q^P \cdot m^O$.
- If m is an O-answer then $s \cdot m = s' \cdot q^P \cdot s'' \cdot m^O$ where m is justified by q. We then have $?(\ulcorner s \cdot m \urcorner) = ?(\ulcorner s' \urcorner qa) = ?(\ulcorner s' \urcorner)$ and since s is well-bracketed, we have $?(s) = ?(s')$. The induction hypothesis permits us to conclude.
- If m is a P-question then $\ulcorner s \cdot m \urcorner = \ulcorner s \urcorner \cdot m$ and $?(\ulcorner s \cdot m \urcorner) = ?(\ulcorner s \urcorner) \cdot m$. Moreover $\ulcorner ?(s \cdot m) \urcorner = \ulcorner ?(s) \cdot m \urcorner = \ulcorner ?(s) \urcorner \cdot m$. By alternation if s is not empty it must end with an O-move so we can conclude using the induction hypothesis.

(ii) The argument is similar to (i). ☐

Note that in (i) and (ii), it is important that s does not end with a P-answer. For instance consider the legal position

$$s = q_0^O\ q_1^P\ q_2^O\ q_3^P\ q_4^O\ a^P$$

ending with a P-answer. We have $\ulcorner ?(s) \urcorner = \ulcorner q_0 \cdot q_1 \cdot q_2 \cdot q_3 \urcorner = q_0 \cdot q_1 \cdot q_2 \cdot q_3$ but $?(\ulcorner s \urcorner) = ?(q_0 \cdot q_1 \cdot q_4 \cdot a) = q_0 \cdot q_1 \cdot q_4$.

By the previous remark and lemma we obtain the following corollary:

Corollary 2.73. *Let $s \cdot m$ be a legal well-bracketed position.*

(i) If m is a P-move then it points in $?(\ulcorner s \urcorner) = \ulcorner ?(s) \urcorner$.

(ii) If m is a non initial O-move then it points in $?(\llcorner s \lrcorner) = \llcorner ?(s) \lrcorner$.

Definition 2.74 (Order)**.** Let $\langle M, \lambda, \vdash \rangle$ be a game. The ***order*** of a question move $q \in M$, written $\operatorname{ord} q$, is given by the length (l) of the longest enabling chain of question moves starting from q ($q = q_1 \vdash q_2 \vdash \ldots \vdash q_l$) minus one (*i.e.*, $\operatorname{ord} q = l - 1$); the order of an answer move is defined as -1. The order of a game $\langle M, \lambda, \vdash \rangle$, written $\operatorname{ord}\langle M, \lambda, \vdash \rangle$, is defined as $\max_{m \in M} \operatorname{ord} m$ with the convention $\max \emptyset = -1$.

For instance the initial question in the game $\mathbb{N}$ has order 0.

Proposition 2.75 (Pointers are superfluous up to order 2)**.** *Let A be a game of order at most 2 where each question move enables at least one answer move (Therefore an order-0 move is necessarily a question enabling answer moves only). Let s be a justified sequence of moves in the game A satisfying alternation, visibility, well-openedness and well-bracketing. If s contains a single initial move then the pointers of the sequence s can be uniquely reconstructed from the underlying sequence of moves.*

Proof. Let A be an arena of order 2 at most and let s be a legal well-bracketed position in L_A. W.l.o.g. we can assume that the game A has a single initial move q_0. Indeed, since s is well-opened, its first move m_0 is the only initial move in the sequence, thus m_0 is the root of some sub-arena A' of A. Hence s can be seen as a play on the game A' instead of A.

Since A is of order 2 at most, all the moves in s except q_0 are of order 1 at most. We prove by induction on the length of s that $?(s)$ corresponds to one of the cases 0, A, B, C, D shown on the table below, and that the pointers in s can be recovered uniquely. Let L denote the language $L = \{\, pq \mid q_0 \vdash p \vdash q \wedge \operatorname{ord} p = 1 \wedge \operatorname{ord} q = 0 \}$.

Case	$\lambda_{OP}(m)$	$?(s) \in$	where...
0	O	$\{\epsilon\}$	
A	P	q_0	
B	O	$q_0 \cdot L^* \cdot p$	$q_0 \vdash p, \operatorname{ord} p = 1$
C	P	$q_0 \cdot L^* \cdot pq$	$q_0 \vdash p \vdash q, \operatorname{ord} p = 1, \operatorname{ord} q = 0$
D	O	$q_0 \cdot L^* \cdot q$	$q_0 \vdash q, \operatorname{ord} q = 0$

Base cases: If s is the empty play then there is no pointer to recover and s corresponds to case 0. If s is a singleton then it must be the initial question q_0, so there is no pointer to recover. This corresponds to case A.
Step case: If $s = u \cdot m$ for some non empty legal well-bracketed position u and move $m \in M_A$ then by the induction hypothesis the pointers in u can all be recovered and u corresponds to one of the cases 0, A, B, C or D. We proceed by case analysis:

case 0 $?(u) = \epsilon$. By Corollary 2.73, m points in $\ulcorner ?(u) \urcorner = \epsilon$. Hence this case is impossible.

case A $?(u) = q_0$ and the last move m is played by P. By Corollary 2.73, m points to q_0. If m is an answer to the initial question q_0 then s is a complete play and $?(s) = \epsilon$, which corresponds to case 0. If m is a first order question then $?(s) = q_0 p$ and it is O's turn to play after s therefore s falls into category B. If m is an order 0 question then s falls into category D.

case B $?(u) \in q_0 \cdot L^* \cdot p$ where $\operatorname{ord} p = 1$ and m is an O-move. By Corollary 2.73, m points in $\ulcorner ?(u) \urcorner = q_0 p$. Since m is an O-move it can only point to p. If m is an answer to p then $?(s) = ?(u \cdot m) \in q_0 \cdot L^*$ which is covered by case A and C. If m is an order 0 question pointing to p then we have $?(s) = ?(u) \cdot m \in q_0 \cdot L^* \cdot pm$ and s falls into category C.

case C $?(u) \in q_0 \cdot L^* \cdot pq$ where $\operatorname{ord} p = 1$, $\operatorname{ord} q = 0$, q_0 justifies p, p justifies q and m is played by P. (i) Suppose that m is an answer, then the well-bracketing condition implies that q is answered first. The move m therefore points to q and we have $?(s) = ?(u \cdot m) \in q_0 \cdot L^* \cdot p$. This corresponds to case B. (ii) Suppose that m is a question, then it is a P-move and therefore is cannot be justified by p. It cannot be justified by q either because q is an order 0 question and therefore enables answer moves only. Similarly m is not justified by any move in L^*. Hence m must point to the initial question q_0. There are two sub-cases, either m is an order 0 move and then s falls into category D or m is an order 1 move and s falls into category B.

case D $?(u) \in q_0 \cdot L^* \cdot q$ where $\operatorname{ord} q = 0$ and m is played by O. Again by Corollary 2.73, m points in $\llcorner ?(u) \lrcorner = q_0 q$. Since m is a P-move it can only point to q. Since q is of order 0, it only enables answer moves therefore m is an answer to q. Hence $?(s) = ?(u \cdot m) \in q_0 \cdot L^*$ and s falls either into category A or C. $\square$

Consequently for order-2 games, plays are entirely determined by the underlying pointer-less sequence of moves. At order 3, however, eliminating pointers causes ambiguities. Take for instance the game $((\mathbb{N}^1 \to \mathbb{N}^2) \to \mathbb{N}^3) \to \mathbb{N}^4$ and sequence of moves $s = q^4 q^3 q^2 q^3 q^2 q^1$, where the superscripts indicate the component of the game in which each move is played. What are the valid plays whose underlying sequence of moves is s? By the visibility condition, the pointers of the first five moves are uniquely determined:

$$s = q^4\ q^3\ q^2\ q^3\ q^2\ q^1\ .$$

For the last move, however, there is an ambiguity: its justifier can be any of the two occurrences of q^2. The visibility condition does not eliminate this ambiguity since both occurrences of q^2 appear in the P-view $\ulcorner s \urcorner = s$. These two possibilities correspond to two different strategies for the Proponent.

2.3.7 Algorithmic game semantics

Game semantics has proved to be a very successful paradigm in fundamental computer science. Following the resolution of the full abstraction problem for PCF, game semantics was subsequently used to obtain fully abstract models of a variety of programming languages. More recently, game semantics has emerged as a new approach to program verification and program analysis. Ghica and McCusker identified a fragment of Idealized Algol for which the game denotation of programs can be expressed using regular expressions. Consequently, the observational equivalence problem for this fragment is decidable [GM00, GM03]. This development opened up a new branch of research called *Algorithmic game semantics* which has interesting applications in program verification [AGOM03, DGL05]. This section gives a quick overview of some important results in the field.

2.3.7.1 Effective presentability

The starting point of *algorithmic game semantics* is a result by Abramsky and McCusker called the Characterization Theorem [AM97, Theorem 25]. We say that a play is *complete* if it is maximal and all questions have been answered. One can show that for every IA type T, the complete plays on the game $[\![T]\!]$ are precisely those in which the initial question has been answered. A game satisfying this condition is said to be *simple* [AM97]. The characterization theorem can then be stated as follows:

Theorem 2.76 (Characterization Theorem for simple games (Abramsky, McCusker [AM97])). *Let σ and τ be strategies on a simple game A. Then:*

$$\sigma \leq \tau \iff comp(\sigma) \subseteq comp(\tau) \ .$$

Thus in the game model of Idealized Algol, observational equivalence is characterized by equality of the set of complete plays.

This result implies that the fully abstract model of Idealized Algol is *effectively presentable* [Loa98b] (*i.e.*, the denotation of a term can be computed by a Turing Machine). The proof crucially relies on the presence of imperative features in IA. Indeed, Loader has shown that even on compact strategies, observation equivalence of PCF is undecidable [Loa01], and this implies that there is no fully abstract model of PCF that is effectively representable.

Algorithmic game semantics is concerned with deriving decision procedures for the observational equivalence problem for various fragments of IA. This problem can be stated as follows: *Given two β-normal forms M and N in a given fragment of IA, does $M \cong N$ hold?* By the Characterization Theorem 2.76, this problem reduces to comparing the set of complete plays of two given terms. Observational equivalence is undecidable in the general case, but it becomes decidable when restricted to some lower-order fragments of IA. This question has now been fully investigated and there is now a complete classification of decidability results for the finitary fragments of IA.

2.3.7.2 The order-2 fragment of IA

Ghica and McCusker were the first to show that the observational equivalence problem becomes decidable when restricting the language IA to some finitary fragment. They showed that for the second-order finitary fragment of Idealized Algol, written IA_2, the set of complete plays of the strategy denotation can be expressed as an extended regular expression [GM00]:

Lemma 2.77 (Ghica and McCusker, [GM00]). *For every IA_2-term $\Gamma \vdash M : T$, the set of complete plays of $[\![\Gamma \vdash M : T]\!]$ is regular.*

Since equivalence of regular expressions is decidable with complexity PSPACE, by the Characterization Theorem this gives a decision procedure for observational equivalence of IA_2-terms. In the same paper they show that the same result holds for the $\text{IA}_2 + \mathsf{while}$ fragment. At order 2, this result cannot be extend further as Ong showed that observational equivalence is already undecidable for $\text{IA}_2 + \mathsf{Y}_1$ [Ong02].

2.3.7.3 Other fragments of IA

Other finitary fragments were subsequently considered. Ong considered the order-3 finitary fragment, denoted IA_3. He showed that the set of complete plays is a context-free language, thus observational equivalence reduces to the *Deterministic Pushdown Automata Equivalence* (DPDA) problem [Ong02]. This problem was shown to be decidable [Sén01] but its complexity is still unknown; we only know that it is primitive recursive [Sti02].

Even for $\text{IA}_3 + \mathsf{while}$, the fragment obtained by throwing in iteration, the problem remains decidable. Moreover the problem lies in EXPTIME [MW05]. For the fragments $\text{IA}_i + \mathsf{Y}_0$ for

$i = 1, 2, 3$, observational equivalence is as difficult as DPDA equivalence (*i.e.*, there is a reduction in both directions) [MOW05]. Finally, Murawski showed that the problem becomes undecidable beyond order 3 (IA_i with $i \geq 4$) [Mur03, Mur05a].

The complete classification of complexity results for IA is recapitulated in Table 2.6. Undefined fragments are marked with the symbol $\times$.

Fragment	pure	+while	+Y0	+Y1
IA_0	PTIME	$\times$	$\times$	$\times$
IA_1	coNP	PSPACE	DPDA EQUIV	$\times$
IA_2	PSPACE	PSPACE	DPDA EQUIV	undecidable
IA_3	EXPTIME	EXPTIME	DPDA EQUIV	undecidable
$\text{IA}_i, i \geq 4$	undecidable	undecidable	undecidable	undecidable

Table 2.6: The complete complexity classification for observational equivalence in IA.

The coNP and PSPACE results are due to Murawski [Mur05b].

Chapter 3

The Safe Lambda Calculus

The *safety* constraint was originally introduced as a syntactical restriction in order to study decidability of Monadic Second Order theories over infinite trees generated by higher-order recursion schemes [KNU02]. The good algorithmic properties of safety in the setting of higher-order recursion schemes (see background chapter) motivate further investigations in the more general setting of the simply-typed lambda calculus. This chapter presents a generalization of the safety syntactic restriction to the lambda calculus, giving rise to what we call the "*safe lambda calculus*".

The first part introduces the typing system of the safe lambda calculus. As remarked in the background chapter, a higher-order grammar can be viewed as a closed simply-typed lambda-term; however this term has a particular shape owing to the structure of the grammatical rules: the right-hand side of a rule is an *applicative* term (*i.e.*, containing no lambda abstraction) of ground type. An adaptation of safety to the lambda calculus setting, however, ought to handle all possible terms, including those containing lambda-abstraction. Our notion of safety is defined in such a way.

The typing system of the safe lambda calculus is a small variation of the simply-typed lambda calculus where the abstraction rule is able to abstract more than one variable at a time but with an extra constraint: the free variables in the resulting term must have order greater than the term itself. The application rule is similarly constrained. The connection with safe higher-order grammars is then made evident by restricting our calculus to pure applicative term: an applicative term of ground type is typable in the safe lambda calculus if and only if it is safe in the sense of Knapik et al.

We study how terms of this language behave with respect to the term conversions commonly studied in the lambda calculus: we adapt the notion of beta-reduction to ensure that a version of the context-reduction lemma holds—safe terms reduce to safe terms—and we show that the conversion to eta-long normal form preserves safety.

Next, in an attempt to quantify the impact of the safety constraint, we look at the complexity of the beta-equivalence problem—Given two safe terms, are they beta-equivalent? The problem is known to be non-elementary for unrestricted terms [Sta79b]. We show PSPACE-hardness for the safe case by reduction from the True Quantifier Boolean Formula problem (TQBF). This PSPACE-complete problem is encodable in the order-3 fragment of the simply-typed lambda calculus, but our encoding in the safe lambda calculus makes use of the entire type-hierarchy. We conjecture the problem to be elementary recursive.

The loss of expressivity caused by safety is then characterized in terms of the numeric functions that are representable: we show that they are precisely the multivariate polynomials *without* the conditional operator. We then give a similar characterization in terms of word functions representable.

We then consider classical typing problems in the setting of the safe lambda calculus: we show that type-checking and typability are decidable and we observe that type inhabitation is (at least) semi-decidable.

To conclude we consider extensions to the simply-typed lambda calculus such as recursion and imperative features. We define the notion of safety in the context of the languages PCF and Idealized Algol.

Remark 3.1 (Related work) A first attempt to adapt the safety restriction to the lambda calculus was made by Aehlig et al. in an unpublished technical report [AdMO04]. The calculus that we present here is both simpler (the typing system is just a slight variation of the simply-typed lambda calculus) and more general (no condition is imposed on types and use of Σ-constants of any order is allowed).

3.1 Definition and properties

3.1.1 Safety adapted to the lambda calculus

We use sequents of the form $\Gamma \vdash_{\$} M : A$ to represent term-in-context where Γ is a typing-context (a consistent set of typing assumptions), A is the type and M is a term (either annotated or untyped). As defined in Sec. 2.1, we write Λ for the set of untyped lambda-terms and $\Lambda_{\mathbb{T}}$ for the set of lambda-terms annotated with simple types $\mathbb{T}$. We will introduced various subscripts \$ to represent terms-in-context from different typing systems. The subscript 'st' refers to the (Curry-style or Church-style) simply-typed lambda calculus (see Convention 3.4).

We fix an atomic type symbol o and for every natural number $n \in \mathbb{N}$ we use *type* notation n to refer to the type n_o defined in Sec. 2.1.5 ($0 \equiv o$ and $(k+1) \equiv k \to o$ for $k \geq 0$). A type $A_1 \to \cdots \to A_n \to B$, where B is not necessarily ground, will be written $(A_1, \cdots, A_n, B)$.

Definition 3.2 (The safe lambda calculus)**.**

(i) The ***safe lambda calculus*** *à la* Curry, denoted "safe $\Lambda_{\to}^{\mathrm{Cu}}$", is a sub-system of the simply-typed lambda calculus *à la* Curry. It is defined as the set of judgements of the form $\Gamma \vdash_{\mathsf{s}} M : A$, where M ranges over untyped term, that are derivable from the system of rules of Table 3.1.

(ii) The ***safe lambda calculus*** *à la* Church, denoted "safe $\Lambda_{\to}^{\mathrm{Ch}}$", is the typing system obtained by adding type annotations in the λ-binders in the abstraction rule of the safe lambda calculus *à la* Curry (see Sec. 2.1.7). In this system, M ranges over *annotated* term.

(iii) The sub-systems defined by the same rules in (i) and (ii), such that all types that occur in them are homogeneous (Sec. 2.2.2), are called the ***homogeneous safe lambda calculus*** *à la* Curry and *à la* Church respectively.

We will consider extension of the safe lambda calculus with constants. For every set Ξ of higher-order constants, we introduce sequents of the form $\Gamma \vdash^{\Xi}_{\$} M : A$, for some subscript \$, to denote the typing system obtained by adding the rule:

$$(\mathsf{const})\ \frac{}{\vdash^{\Xi}_{\$} f : A}\ f \in \Xi\ .$$

For convenience, we shall omit the superscript from $\vdash^{\Xi}_{\$}$ whenever the set of constants Ξ is clear from the context.

The safe lambda calculus deviates from the standard definition of the simply-typed lambda calculus in a number of ways. First the application and abstraction rules can respectively perform multiple applications and abstract several variables at once. (Of course this feature alone does not alter expressivity.) Crucially, the side-conditions in the application rule and abstraction rule require the variables in the typing context to have orders no smaller than that of the term being formed. Safe terms can be applied together using the rule ($\mathsf{app}_{\mathsf{as}}$), but the resulting term is only "almost-safe"; it can be turned into a safe term by subsequently applying the abstraction

$$(\mathsf{var})\ \frac{}{x : A \vdash_{\mathsf{s}} x : A} \qquad (\mathsf{wk})\ \frac{\Gamma \vdash_{\mathsf{s}} M : A}{\Delta \vdash_{\mathsf{s}} M : A}\ \Gamma \subset \Delta \qquad (\delta)\ \frac{\Gamma \vdash_{\mathsf{s}} M : A}{\Gamma \Vdash_{\mathsf{app}} M : A}$$

$$(\mathsf{app_{as}})\ \frac{\Gamma \vdash_{\mathsf{s}} M : (A_1, \ldots, A_n, B) \quad \Gamma \vdash_{\mathsf{s}} N_1 : A_1 \quad \ldots \quad \Gamma \vdash_{\mathsf{s}} N_n : A_n}{\Gamma \Vdash_{\mathsf{app}} M\, N_1 \ldots N_n : B}$$

$$(\mathsf{app})\ \frac{\Gamma \vdash_{\mathsf{s}} M : (A_1, \ldots, A_n, B) \quad \Gamma \vdash_{\mathsf{s}} N_1 : A_1 \quad \ldots \quad \Gamma \vdash_{\mathsf{s}} N_n : A_n}{\Gamma \vdash_{\mathsf{s}} M\, N_1 \ldots N_n : B}\quad \operatorname{ord}\Gamma \geq \operatorname{ord} B$$

$$(\mathsf{abs})\ \frac{\Gamma, x_1 : A_1, \ldots, x_n : A_n \Vdash_{\mathsf{app}} M : B}{\Gamma \vdash_{\mathsf{s}} \lambda x_1 \ldots x_n.M : (A_1, \ldots, A_n, B)}\quad \operatorname{ord}\Gamma \geq \operatorname{ord}(A_1, \ldots, A_n, B)$$

where $\operatorname{ord}\Gamma$ denotes the set $\{\operatorname{ord} A \mid y : A \in \Gamma\}$ and for $S \subseteq \mathbb{N}$, $u \in \mathbb{N}$, "$S \geq u$" means that u is a lower-bound of S.

Table 3.1: The safe lambda calculus *à la* Curry.

rule. We do not impose any constraint on types. In particular, type-homogeneity, which was an assumption of the original definition of safe grammars [KNU02], is not required here. Another difference is that we allow the addition of Ξ-constants with arbitrary higher-order types.

Definition 3.3 (Safe terms)**.**

(i) An *untyped* term $M \in \Lambda$ is ***safe*** if the judgement $\Gamma \vdash_{\mathsf{s}} M : T$ is derivable in the safe lambda calculus *à la* Curry for some context Γ and type T. Otherwise it is said to be ***unsafe***.

(ii) A *type-annotated* term $M \in \Lambda_{\mathbb{T}}$ is ***safe*** if the judgement $\Gamma \vdash_{\mathsf{s}} M : T$ is derivable in the safe lambda calculus *à la* Church for some context Γ and type T. Otherwise it is said to be ***unsafe***.

(iii) An *untyped* term $M \in \Lambda$ is ***universally safe*** if all its valid type annotations are safe (*i.e.*, for every $M' \in \Lambda_{\mathbb{T}}$, context Γ and type A such that $\Gamma \vdash_{\mathrm{Ch}} M' : A$ and $|M'| \equiv M$, M' is safe). It is ***universally unsafe*** if all its valid type annotations are unsafe.

(iv) A term M that is typable as $\Gamma \Vdash_{\mathsf{app}} M : T$ for some Γ, T is called an ***almost safe application***.

(v) A *term-in-context* $\Gamma \vdash_{\mathsf{st}} M : T$ of the Curry-style (resp. $\Gamma \vdash_{\mathrm{Ch}} M : T$ of the Church-style) simply-typed lambda calculus is said to be *safe* if $\Gamma \vdash_{\mathsf{s}} M : T$ is also typable in the Curry-style (resp. Church-style) safe lambda calculus.

Convention 3.4 To avoid cumbersome notations, we will use sequents of the form $\Gamma \vdash_{\mathsf{s}} M : A$ to refer to judgements of both versions of the safe lambda calculus (Curry and Church). When we specify that M is an untyped term in Λ then it is understood that the judgement refers to a term-in-context typed in the Curry-style safe lambda calculus; if M ranges over annotated terms in $\Lambda_{\mathbb{T}}$ then it refers to a term-in-context typed in the Church-style safe lambda calculus. When the domain of M is not specified then it means that the current argument, definition, lemma or proposition is valid in both systems.

Example 3.5 (Kierstead terms)**.** Consider the annotated terms $M_1 \equiv \lambda f^2.f(\lambda x^o.f(\lambda y^o.y))$ and $M_2 \equiv \lambda f^2.f(\lambda x^o.f(\lambda y^o.x))$. M_2 is unsafe because in the subterm $f(\lambda y^o.x)$, the free variable x has order 0 which is smaller than $\operatorname{ord}(\lambda y^o.x) = 1$. On the other hand, M_1 is safe as the following proof tree shows:

$$
\dfrac{\dfrac{\dfrac{}{f : 2 \vdash_{\mathsf{s}} f : 2}\,(\mathsf{var}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{f : 2 \vdash_{\mathsf{s}} f : 2}\,(\mathsf{var})}{f : 2, x : o \vdash_{\mathsf{s}} f : 2}\,(\mathsf{wk}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{y : o \vdash_{\mathsf{s}} y : o}\,(\mathsf{var})}{y : o \Vdash_{\mathsf{app}} y : o}\,(\delta)}{y : o \vdash_{\mathsf{s}} \lambda y^o.y : 1_o}\,(\mathsf{abs})}{f : 2, x : o \vdash_{\mathsf{s}} \lambda y^o.y : 1_o}\,(\mathsf{wk})}{f : 2, x : o \vdash_{\mathsf{s}} f(\lambda y^o.y) : o}\,(\mathsf{app})}{f : 2 \vdash_{\mathsf{s}} \lambda x^o.f(\lambda y^o.y) : (o, o)}\,(\mathsf{abs})}{f : 2 \vdash_{\mathsf{s}} f(\lambda x^o.f(\lambda y^o.y)) : o}\,(\mathsf{app})}{\vdash_{\mathsf{s}} M_1 \equiv \lambda f^2.f(\lambda x^o.f(\lambda y^o.y)) : 3}\,(\mathsf{abs})
$$

Now consider the untyped terms underlying M_1 and M_2: $|M_1| \equiv \lambda f.f(\lambda x.f(\lambda y.y))$ and $|M_2| \equiv \lambda f.f(\lambda x.f(\lambda y.x))$ both have for principal type $\alpha_3 \equiv ((\alpha \to \alpha) \to \alpha) \to \alpha$. Further, every typing derivation for $|M_1|$ and $|M_2|$ in the simply-typed lambda calculus assigns the same type α to the occurrences of the variables x and y. Hence $|M_1|$ is *universally* safe and $|M_2|$ is *universally* unsafe.

Example 3.6. The term-in-context $f : (1, 1, o) \Vdash_{\mathsf{app}} (\lambda \varphi^2 \theta^3.\varphi(\lambda x^o.x))(f(\lambda x^o.x)) \equiv M : 3$ is almost safe. Abstracting f produces the safe term-in-context $\vdash_{\mathsf{s}} \lambda f^{(1,1,o)}.M : ((1, 1, o), 3)$.

The basic properties of the simply-typed lambda calculus also hold in the safe lambda calculus:

Lemma 3.7. *(i)* $\Gamma \vdash_{\mathsf{s}} M : B \ \wedge\ \Gamma \subseteq \Gamma' \implies \Gamma' \vdash_{\mathsf{s}} M : B$

(ii) $\Gamma \vdash_{\mathsf{s}} M : B \implies FV(M) \subseteq \mathrm{dom}(\Gamma)$

(iii) $\Gamma \vdash_{\mathsf{s}} M : B \implies \Gamma_M \vdash_{\mathsf{s}} M : B$ *where* $\Gamma_M = \{z : A \in \Gamma \mid z \in FV(M)\}$.

Proof. Trivial. ☐

It is easy to see that valid typing judgements of the safe lambda calculus satisfy the following simple invariant that we will later refer as the *basic property of the safe lambda calculus*:

Lemma 3.8 (Basic property)**.** *Let* $\Gamma \vdash_{\mathsf{s}} M : B$ *be a valid judgement of the Curry or Church-like safe lambda calculus. Then*

$$\forall z : A \in \Gamma : \ z \in FV(M) \implies \mathrm{ord}\, A \geq \mathrm{ord}\, B \ .$$

Note that the converse does not hold: Take the annotated term $\lambda y^o z^o.(\lambda x^o.y)z$. Since it is closed, it trivially satisfies the condition in the conclusion of the previous lemma, but it is not safe because the variable y is not abstracted by the abstraction 'λx'. The converse does not even hold for applicative terms: for instance the term-in-context $f : 2, g : (o, o, o), y : o \vdash_{\mathsf{st}} f(g\,y) : o$ satisfies the condition of the lemma but it is unsafe because the term $g\,y$ of type 1 occurs in operand position and contains a free occurrence of a ground-type variable y.

Subterms

The Subterm Lemma of the simply-typed lambda calculus does not hold anymore: a safe term may contain unsafe subterms. For instance the term $\lambda fx.fx$ is universally safe however its subterm $\lambda x.fx$ is universally unsafe. There is, however, a subclass of subterms for which this result holds:

Definition 3.9 (Large subterms)**.** Let M be an untyped term, the set $\widetilde{\mathrm{sub}}(M)$ of ***large subterms*** of M is defined inductively by

$$
\begin{aligned}
\widetilde{\mathrm{sub}}(x) &= \{x\} \\
\widetilde{\mathrm{sub}}(MN) &= \{N\} \cup \widetilde{\mathrm{sub}}(M) \cup \widetilde{\mathrm{sub}}(N) \\
\widetilde{\mathrm{sub}}(\lambda\overline{x}.M) &= \{\lambda\overline{x}.M\} \cup \widetilde{\mathrm{sub}}(M) \text{ where } M \text{ is not an abstraction.}
\end{aligned}
$$

The set of large subterms of an annotated type is defined identically.

Lemma 3.10 (Subterm lemma for safe $\Lambda^{\mathrm{Ch}}_{\rightarrow}$ and safe $\Lambda^{\mathrm{Cu}}_{\rightarrow}$)**.** *Let M range over Λ or $\Lambda_{\mathbb{T}}$. Then*

$$\Gamma \vdash_{\mathsf{s}} M : T \ \wedge\ M' \in \widetilde{\mathrm{sub}}(M) \implies \Gamma' \vdash_{\mathsf{s}} M' : T' \textit{ for some } \Gamma', T'.$$

Proof. The proof is a trivial induction on the structure of the term □

To indicate that a term is unsafe we will sometimes highlight the source of its unsafety by underlining one of its large subterm as well as some free occurrence of a variable in that subterm that does not satisfy the condition of the previous Lemma; we may underline just the variable if the large subterm is clear. For instance the term $\lambda f^2.f(\lambda x^o.f(\underline{\lambda y^o.\underline{x}}))$ is unsafe because the subterm $\lambda y^o.x$ has order greater than the order of the variable x occurring free in it.

The applicative homogeneously-typed fragment of the safe lambda calculus captures the original notion of safety due to Knapik et al. in the context of higher-order grammars (Def. 2.32):

Proposition 3.11 (Correspondence with safe grammars)**.** *Let $G = \langle \Sigma, \mathcal{N}, \mathcal{R}, S\rangle$ be a grammar and let e be an applicative term generated from the symbols in $\mathcal{N} \cup \Sigma \cup \{ z_1^{A_1}, \cdots, z_m^{A_m} \}$. A rule $F z_1 \ldots z_m \rightarrow e$ in $\mathcal{R}$ is safe (in the original sense of Knapik et al.) if and only if $z_1 : A_1, \cdots, z_m : A_m \vdash_{\mathsf{s}}^{\Sigma\cup\mathcal{N}} e : o$ is a valid typing judgement of the* homogeneous *(Curry or Church-style) safe lambda calculus.*

Proof. First we observe that since e is an applicative term, the distinction between Curry and Church-style lambda calculus does not matter. We show by induction that

(i) $z_1, \ldots, z_m \Vdash_{\mathsf{app}} t : A$ is a valid judgement of the homogeneous safe lambda calculus containing no abstraction if and only if in the Knapik sense, all the occurrences of unsafe subterms of t are safe occurrences.

(ii) $z_1, \ldots, z_m \vdash_{\mathsf{s}} t : A$ is a valid judgement of the homogeneous safe lambda calculus containing no abstraction if and only if in the Knapik sense, all the occurrences of unsafe subterms of t are safe occurrences, and all parameters occurring in t have order greater than $\mathrm{ord}\, t$.

The constant and variable rules are trivial. Application case: By definition, a term $t_0 \ldots t_n$ is Knapik-safe iff for all $0 \leq i \leq n$, all the occurrences of unsafe subterms of t_i are safe occurrences (in the Knapik sense), and for all $1 \leq j \leq n$, the operands occurring in t_j have order greater than $\mathrm{ord}\, t_j$. The $(\mathsf{app}_{\mathsf{as}})$ rule and the induction hypothesis permit us to conclude.

Now since e is an applicative term of *ground type*, the previous result gives: $z_1, \ldots, z_m \vdash_{\mathsf{s}} e : o$ is a valid judgement of the homogeneous safe lambda calculus iff all the occurrences of unsafe subterms of e are safe occurrences, which by definition of Knapik-safety is equivalent to saying that the rule $F z_1 \ldots z_m \rightarrow e$ is safe. □

Remark 3.12 This result was first proved by de Miranda [dM06] for a different notion of safe lambda calculus. See Remark 3.53.

In what sense is the safe lambda calculus *safe*?

It is an elementary fact that when performing β-reduction in the lambda calculus, one must use capture-*avoiding* substitution, which is standardly implemented by renaming bound variables afresh upon each substitution. In the safe lambda calculus, however, variable capture can never happen (as the following lemma shows). Substitution can therefore be implemented simply by capture-*permitting* replacement, without any need for variable renaming.

Convention 3.13 (Safe variable typing convention) We say that a set Γ of typing assumptions of the form $x : A$, for some variable x and simple type T, is ***order-consistent*** if all the types assigned to a given variable are of the same order:

$$x : A_1 \in \Gamma \ \wedge \ x : A_2 \in \Gamma \implies \operatorname{ord} A_1 = \operatorname{ord} A_2 \ .$$

Let $M \in \Lambda_{\mathbb{T}}$ be an annotated term. We define the set $Ass(M)$ as the set of type-assignments induced by the type annotations in M:

$$\begin{aligned} Ass(x) &= \emptyset \\ Ass(M\,N) &= Ass(M) \cup Ass(N) \\ Ass(\lambda x^T.M) &= \{x : T\} \cup Ass(M) \ . \end{aligned}$$

By extension, the set of type-assignments induced by a term-in-context $\Gamma \vdash_{\mathrm{Ch}} M : T$ is given by $Ass(\Gamma \vdash_{\mathrm{Ch}} M : T) = \Gamma \cup Ass(M)$. A type-annotated term M is said to be ***order-consistent*** just if the set $Ass(M)$ is; a countable set of terms $M_0, M_1, \ldots$ is ***order-consistent*** just if $\bigcup_{i\geq 0} Ass(M_i)$ is. This notion naturally extends to (countable sets of) terms-in-context.

We now adopt the ***safe variable typing convention***: In any definition, theorem or proof involving countably many terms, it is assumed that the set of terms involved is order-consistent.

Example 3.14. The set of typing assumptions $\{x : o, x : 1\}$ is not order-consistent. Therefore the annotated term $\lambda x^1.x(\lambda x^o.x)$ is not order-consistent; however, it is alpha-equivalent to the term $\lambda y^1.y(\lambda x^o.x)$ which is order-consistent.

The set of terms $\{\lambda x^0.x, \lambda x^1.x\}$ is not order-consistent.

In the following, we write $M\,\{N/x\}$ to denote the capture-*permitting* substitution that textually replaces all free occurrences of x in M by N without performing variable renaming (see Def. 2.4) and $M\left\{\overline{N}/\overline{x}\right\}$ to refer to its simultaneous variant (Def. 2.6).

Lemma 3.15 (No-variable-capture lemma)**.** *In the safe lambda calculus à la Church, there is no variable capture when performing simultaneous capture-permitting substitution provided that we adopt the* safe variable typing convention *(Convention 3.13): If* $\Gamma, \overline{x} : \overline{B} \vdash_{\mathsf{s}} M : A$, $\Gamma \vdash_{\mathsf{s}} N_1 : B_1, \cdots, \Gamma \vdash_{\mathsf{s}} N_n : B_n$, *where* $|\overline{x}| = n$ *then*

$$M\{\overline{N}/\overline{x}\} \equiv M[\overline{N}/\overline{x}] \ .$$

Proof. We prove the result by structural induction on M. The variable and constant cases are trivial. Otherwise, M is of the form $\lambda\overline{y}^{\overline{C}}.M_0 \ldots M_m$ where $\overline{y} = y_1 \ldots y_p$, $m, p \geq 0$ and for every $0 \leq i \leq m$, M_i is safe. The simultaneous capture-permitting substitution gives:

$$M\left\{\overline{N}/\overline{x}\right\} \equiv \lambda\overline{y}^{\overline{C}}.M_0\left\{\overline{N}\upharpoonright I/\overline{x}\upharpoonright I\right\} \ldots M_m\left\{\overline{N}\upharpoonright I/\overline{x}\upharpoonright I\right\}$$

where $I = \{i \in 1..n \mid x_i \notin \overline{y}\}$ and for every list s, $s \upharpoonright I$ denotes the sublist of s obtained by keeping only elements in s whose position index in the list belongs to I.

Suppose for contradiction that a variable capture occurs in $M\left\{\overline{N}/\overline{x}\right\}$. By the induction hypothesis there is no variable capture in $M_i\left\{\overline{N}\upharpoonright I/\overline{x}\upharpoonright I\right\}$ for $0 \leq i \leq m$. This means that we are in the following situation: For some $i \in I$ and $1 \leq j \leq p$ the variable y_j occurs freely in N_i, and x_i occurs freely in M. Since $y_j \in FV(N_i)$ we must have $y_j : D \in \Gamma$ for some type D, and by the *safe variable typing convention*, we necessarily have $\operatorname{ord} D = \operatorname{ord} C_j$. Therefore:

$$\begin{aligned} \operatorname{ord} D &\geq \operatorname{ord} B_i && \text{by Lemma 3.8 since } y_j \in FV(N_i), \\ &\geq \operatorname{ord} A && \text{by Lemma 3.8 since } x_i \in FV(M), \\ &= 1 + \max\{\operatorname{ord} C_k \mid 1 \leq k \leq p\} \\ &> \operatorname{ord} C_j \\ &= \operatorname{ord} D && \text{by the \textit{safe variable typing} convention,} \end{aligned}$$

which gives us a contradiction. ☐

Example 3.16. (i) In order to contract the β-redex in

$$f:(o,o,o), x:o \vdash_{\mathrm{Ch}} (\lambda \varphi^{(o,o)} x^o.\varphi\, x)(\underline{f\, x}):(o,o)$$

one should rename afresh the bound variable x to prevent the capture of the free occurrence of x in the underlined subterm during substitution. Consequently, by the previous lemma, the term is not safe. And indeed the basic property of the safe lambda calculus is not satisfied because $\operatorname{ord} x = 0 < 1 = \operatorname{ord} f x$.

(ii) Adopting the safe variable typing convention is crucial for the lemma to hold. For instance take the safe terms $M \equiv \lambda y^o.x$ and $N \equiv y$. We have $x:1 \vdash_{\mathsf{s}} M : o \rightarrow 1$ and $y:1 \vdash_{\mathsf{s}} N : 1$. But

$$M\{N/x\} \equiv \lambda y^o.y \not\equiv \lambda x^o.y \equiv M[N/x] \enspace .$$

Alternatively, the following version of the No-variable capture Lemma does not rely on Convention 3.13:

Lemma 3.17. *Let $\Gamma, \overline{x} : \overline{B} \vdash_{\mathsf{s}} M : A$, $\Gamma \vdash_{\mathsf{s}} N_1 : B_1, \cdots, \Gamma \vdash_{\mathsf{s}} N_n : B_n$, with $|\overline{x}| = n$, be valid judgements of the safe lambda calculus à la Church. Then if further $\Gamma \vdash_{\mathrm{Ch}} M\{\overline{N}/\overline{x}\} : A$ is a valid Church simply-typed term-in-context (not-necessarily safe) then*

$$M\{\overline{N}/\overline{x}\} \equiv M[\overline{N}/\overline{x}] \enspace .$$

Proof. The proof is the same as for the previous Lemma except that to show that $\operatorname{ord} C_j = \operatorname{ord} C$ we use the assumption $\Gamma \vdash_{\mathrm{Ch}} M\{\overline{N}/\overline{x}\} : A$ instead of the safe typing convention: Since the annotated term $\lambda \overline{y}^{\overline{C}}.M_0 \left\{\overline{N} \upharpoonright I/\overline{x} \upharpoonright I\right\} \ldots M_m \left\{\overline{N} \upharpoonright I/\overline{x} \upharpoonright I\right\}$ is typable in the Church-like lambda calculus, the free variables y_j in N_i must be bound by the abstraction $\lambda \overline{y}^{\overline{C}}$. Consequently its type must be C_j. Hence $D \equiv C_j$ and $\operatorname{ord} D = \operatorname{ord} C_j$. ☐

Remark 3.18 A version of the No-variable-capture Lemma also holds in safe grammars, as is implicit in (for example Lemma 3.2 of) the original paper [KNU02].

Note that lambda-terms that do not require variable-capture when being reduced are not necessarily safe. For instance the β-redex in $\lambda y^o z^o.(\lambda x^o.y)z$ can be soundly contracted using capture-permitting substitution, even though the term is not safe.

Lemma 3.19 (Substitution Lemma)**.** *Let $\Gamma \vdash_{\mathsf{s}} N : A$. Then*

(i) $\Gamma, x:A \vdash_{\mathsf{s}} M : B \implies \Gamma \vdash_{\mathsf{s}} M[N/x] : B$,

(ii) $\Gamma, x:A \Vdash_{\mathsf{app}} M : B \implies \Gamma \Vdash_{\mathsf{app}} M[N/x] : B$.

Further if $\Gamma \vdash_{\mathsf{s}} N : A$ and $\Gamma \vdash_{\mathsf{s}} M : A$ are homogeneously safe then so is $\Gamma \vdash_{\mathsf{s}} M[N/x] : B$, and if $\Gamma \vdash_{\mathsf{s}} N : A$ and $\Gamma \vdash_{\mathsf{s}} M : A$ are homogeneously almost-safe then so is $\Gamma \vdash_{\mathsf{s}} M[N/x] : B$.

Proof. Let $\Gamma \vdash_{\mathsf{s}} N : A$. We show (i) and (ii) simultaneously by induction on the derivation tree of $\Gamma, x:A \vdash_{\mathsf{s}} M : B$ or $\Gamma, x:A \Vdash_{\mathsf{app}} M : B$. The base cases (var) and (const) are trivial. The cases (δ) and (wk) follow immediately from the induction hypothesis.

Case (abs): We have $\Gamma, x:A \vdash_{\mathsf{s}} \lambda \overline{y}^{\overline{C}}.Q \equiv M : (\overline{C}, D)$. Suppose that x belongs to $\overline{y}$ then the substitution is not pushed inside the lambda so the result holds trivially. Otherwise suppose that $\Gamma, x:A, \overline{y}:\overline{C} \Vdash_{\mathsf{app}} Q : D$. Applying the induction hypothesis (ii) on this term-in-context gives: $\Gamma, \overline{y}:\overline{C} \Vdash_{\mathsf{app}} Q[N/x] : D$ and by the rule (abs) we obtain: $\Gamma \vdash_{\mathsf{s}} \lambda \overline{y}^{\overline{C}}.Q[N/x] : (\overline{C}, D)$. We can then conclude since $\lambda \overline{y}^{\overline{C}}.Q[N/x] \equiv (\lambda \overline{y}^{\overline{C}}.Q)[N/x]$ under the *safe variable naming convention* (Convention 3.13).

Case ($\mathsf{app_{as}}$): We have $M \equiv M_0 M_1 \ldots M_p$ for $p \geq 1$ and $\Gamma \vdash_{\mathsf{s}} M_k : A_k$ for $1 \leq k \leq p$. By the induction hypothesis, we have $\Gamma \vdash_{\mathsf{s}} M_k\,[N/x] : A_k$ for all k. The rules ($\mathsf{app_{as}}$) permit us to conclude.

Case (app): Again it is proved by applying the induction hypothesis on the premises of the rules.

Finally, term substitution preserves types so in particular it preserves type homogeneity. □

REMARK 3.20 (i) This result naturally extends to simultaneous substitution: If $\Gamma \vdash_{\mathsf{s}} N_k : A_k$ for $1 \leq k \leq n$ then $\Gamma, x_1 : A_1, \ldots x_n : A_n \vdash_{\mathsf{s}} M : B$ implies $\Gamma \vdash_{\mathsf{s}} M[N_1/x_1, \ldots, N_n/x_n] : B$ and $\Gamma, x_1 : A_1, \ldots x_n : A_n \Vdash_{\mathsf{app}} M : B$ implies $\Gamma \Vdash_{\mathsf{app}} M[N_1/x_1, \ldots, N_n/x_n] : B$.

(ii) Observe that the *type* substitution lemma of the simply-typed lambda calculus does not hold in the safe lambda calculus. This is because type substitution allows one to alter the order of the variables occurring in the term. For instance take $M \equiv \lambda f y.f(\lambda x.y)$. Its principal type in the lambda calculus is $A \equiv ((\alpha \to \beta) \to \gamma) \to \beta \to \gamma$ for some atomic types α, β and γ. Then the judgement $\vdash_{\mathsf{st}} M : A$ is unsafe (because $\operatorname{ord} y = \operatorname{ord} x$), the judgement $\vdash_{\mathsf{st}} M : A\,[\beta \to \beta/\beta]$ is safe, and the judgement $\vdash_{\mathsf{st}} M : A\,[\beta \to \beta/\beta]\,[\alpha \to \alpha/\alpha]$ is unsafe.

3.1.2 Safe beta reduction

It is desirable to have an appropriate notion of reduction for our calculus. The standard β-reduction rule is not adequate, however, because safety is not preserved by β-reduction as the following example shows: The safe term $\lambda f^{(o,o,o)} z^o w^o.(\lambda x^o y^o.f x y) z w$ β-reduces in one step to $\lambda f^{(o,o,o)} z^o w^o.(\underline{\lambda y^o.f z y}) w$, which is unsafe since the underlined order-1 subterm contains a free occurrence of a ground variable; but if we perform one more reduction we obtain the safe term $\lambda f^{(o,o,o)} z^o w^o.f z w$. This suggests simultaneous contraction of "consecutive" β-redexes. In order to define this notion of reduction we first introduce the corresponding notion of redex.

In the simply-typed lambda calculus a redex is a term of the form $(\lambda x.M)N$. In the safe lambda calculus, a redex is a succession of several standard redexes:

Definition 3.21 (Safe redex)**.** An ***untyped safe redex*** is an untyped almost safe application of the form $(\lambda x_1 \ldots x_n.M) N_1 \ldots N_l$ for some $l, n \geq 1$ such that M is an almost safe application. (Consequently $\lambda x_1 \ldots x_n.M$ is safe and each N_i is safe for $1 \leq i \leq n$.) The notion of *annotated* safe redex is defined similarly.

For instance, in the case $n < l$, a safe redex has a derivation tree of the following form:

$$
\dfrac{
\mathsf{(wk)}\,\dfrac{\mathsf{(abs)}\,\dfrac{\cdots}{\Gamma', \overline{x} : \overline{A} \vdash_{\mathsf{s}} M : (A_{n+1}, \ldots, A_l, B)}}{\Gamma \vdash_{\mathsf{s}} \lambda \overline{x}.M : (A_1, \ldots, A_l, B)}
\qquad
\dfrac{\cdots}{\Gamma \vdash_{\mathsf{s}} N_1 : A_1}
\qquad
\dfrac{\cdots}{\Gamma \vdash_{\mathsf{s}} N_l : A_l}
}{\Gamma \vdash_{\mathsf{s}} (\lambda \overline{x}.M) N_1 \ldots N_l : B}\,\mathsf{(app)}
$$

where the abbreviations $\overline{x}$ and $\overline{x} : \overline{A}$ stand for $x_1 \ldots x_n$ and $x_1 : A_1, \ldots, x_n : A_n$ respectively.

Example 3.22. The term $(\lambda f^1.((\lambda g^1 h^1.h)(\lambda z^o.z)))\,(\lambda z^o.z)\,(\lambda z^o.z)$ is a safe redex of type $o \to o$. This example shows that there exist safe redexes of the form $(\lambda x_1 \ldots x_n.M) N_1 \ldots N_l$ with $l > n$.

A *safe redex* is by definition an almost safe term, but it is not necessarily a *safe term*. For instance the term $(\lambda x^o y^o.x) z$ is a safe redex but it is only an *almost* safe term. The reason why we call such redexes "safe" is because when they occur within a safe term, it is possible to contract them without breaking the safety of the whole term. Before proving this result, we first define how safe redexes are contracted:

Definition 3.23 (Safe redex contraction)**.** We use the abbreviations $\overline{x} = x_1 \ldots x_n$, $\overline{N} = N_1 \ldots N_l$ and $\overline{y} = y_1 \ldots y_1$m for $n, l, q \geq 1$. The relation β_s (when viewed as a function) is defined on the set of safe redexes as follows:

$$\begin{aligned}\beta_s = \{\ & (\lambda\overline{x}.M)N_1 \ldots N_l \mapsto \lambda x_{l+1} \ldots x_n.M\left[\overline{N}/x_1 \ldots x_l\right] \ \mid n > l\} \\ \cup \{\ & (\lambda\overline{x}.M)N_1 \ldots N_l \mapsto M\left[N_1 \ldots N_n/\overline{x}\right] N_{n+1} \ldots N_l \mid n \leq l\}\ .\end{aligned}$$

where the notation $M\left[R_1 \ldots R_k/z_1 \ldots z_k\right]$ denotes the simultaneous substitution (Def. 2.7).

Lemma 3.24 (β_s preserves safety)**.** *Suppose that $M_1 \, \beta_s \, M_2$. Then*

(i) M_2 is almost safe;

(ii) $\Gamma \vdash_{\mathsf{s}} M_1 : A \implies \Gamma \vdash_{\mathsf{s}} M_2 : A$.

Proof. Let $M_1 \, \beta_s \, M_2$ for some almost safe redex M_1 and term M_2 of type A. By definition, M_1 is of the form $(\lambda x_1 \ldots x_n.M)N_1 \ldots N_l$ for some safe terms $N_1, \ldots, N_l$ of type $B_1, \ldots, B_n$; almost safe term M of type C; and such that $(\lambda x_1 \ldots x_n.M)$ is a safe term of type $(B_1, \ldots, B_n, C)$.

- Suppose $n > l$ then $A = (B_{l+1}, \ldots, B_n, C)$. (i) By the Substitution Lemma 3.19(ii), the term $M\left[\overline{N}/x_1 \ldots x_l\right] : C$ is an almost safe application: $\Gamma, x_{l+1} : B_{l+1}, \ldots x_n : B_n \Vdash_{\mathsf{app}} M\left[\overline{N}/x_1 \ldots x_l\right] : C$. Thus by definition, $\lambda x_{l+1} \ldots x_n.M\left[\overline{N}/x_1 \ldots x_l\right] \equiv M_2$ is almost safe.
  (ii) Suppose that M_1 is safe. W.l.o.g. we can assume that the last rule used to form M_1 is (app) (and not the weakening rule (wk)), thus we have $\operatorname{dom}\Gamma = FV(M_1)$, and Lemma 3.8 gives us $\operatorname{ord} A \leq \operatorname{ord}\Gamma$. This allows us to use the rule (abs) to form the safe term $\Gamma \vdash_{\mathsf{s}} \lambda x_{l+1} \ldots x_n.M\left[\overline{N}/x_1 \ldots x_l\right] \equiv M_2 : A$.
- Suppose $n \leq l$. (i) Again by the Substitution Lemma we have that $M\left[N_1 \ldots N_n/\overline{x}\right]$ is an almost safe application: $\Gamma \Vdash_{\mathsf{app}} M\left[N_1 \ldots N_n/\overline{x}\right] : C$. If $n = l$ then the proof is finished; otherwise $(n < l)$ we further apply the rule $(\mathsf{app}_{\mathsf{as}})$ $l - n$ times which gives us the almost safe application $\Gamma \Vdash_{\mathsf{app}} M_2 : A$.
  (ii) Suppose that M_1 is safe. If $n = l$ then $M_2 \equiv M\left[N_1 \ldots N_n/\overline{x}\right]$ is safe by the Substitution Lemma; if $n < l$ then we obtain the judgement $\Gamma \vdash_{\mathsf{s}} M_2 : A$ by applying the rule $(\mathsf{app}_{\mathsf{as}})$ $l - n - 1$ times on $\Gamma \vdash_{\mathsf{s}} M\left[N_1 \ldots N_n/\overline{x}\right] : C$ followed by one application of (app). ☐

We can now define a notion of reduction for safe terms:

Definition 3.25 (Safe beta-reduction)**.** The ***safe β-reduction***, written $\rightarrow_{\beta_s}$, is the compatible closure of the relation β_s with respect to the formation rules of the safe lambda calculus (*i.e.*, it is the smallest relation such that if $M_1 \, \beta_s \, M_2$ and $C[M]$ is a safe term for some context $C[-]$ formed with the rules of the simply-typed lambda calculus then $C[M_1] \rightarrow_{\beta_s} C[M_2]$). The relation $=_{\beta_s}$ is defined as the reflexive, symmetric, transitive closure of $\rightarrow_{\beta_s}$.

Lemma 3.26. *The safe reduction relation $\rightarrow_{\beta_s}$:*

(i) is a subset of the transitive closure of $\rightarrow_\beta$ $(\rightarrow_{\beta_s} \subset \twoheadrightarrow_\beta)$;

(ii) is strongly normalizing;

(iii) has the unique normal form property;

(iv) has the Church-Rosser property.

Proof. (i) Immediate from the definition: The safe β-reduction is just a multi-step β-reduction. (ii) This is because $\rightarrow_{\beta_s} \subset \twoheadrightarrow_\beta$, and $\rightarrow_\beta$ is strongly normalizing in the simply-typed lambda calculus. (iii) It is easy to see that a safe term has a beta-redex if and only if it has a safe beta-redex (Because a beta-redex can always be "widen" into consecutive beta-redexes of the shape of those in Def. 3.23). Therefore the set of β_s-normal forms is equal to the set of β_s-normal forms. The uniqueness of β-normal form then implies the uniqueness of β_s-normal form. (iv) is a consequence of (i) and (ii). ☐

Since $\rightarrow_{\beta_s}$ is by definition the compatible closure of β_s by the formation rules of the safe lambda calculus, Lemma 3.24 implies

Lemma 3.27 (Subject Reduction)**.** *Let $M_1 \rightarrow_{\beta_s} M_2$. Then*

(i) $\Gamma \vdash_{\mathsf{s}} M_1 : B \implies \Gamma \vdash_{\mathsf{s}} M_2 : B$*,*

(ii) $\Gamma \Vdash_{\mathsf{app}} M_1 : B \implies \Gamma \Vdash_{\mathsf{app}} M_2 : B$*.*

Proof. Suppose that $M_1 \rightarrow_{\beta_s} M_2$. Then we have $M_1 \equiv C[R_1]$ and $M_2 \equiv C[N_2]$ for some context $C[-]$ and safe redex N_1 with $N_1 \, \beta_s \, N_2$.

(i) If the safe redex N_1 is a safe term $\Gamma' \vdash_{\mathsf{s}} N_1 : A$ then by Lemma 3.24(ii) we have $\Gamma' \vdash_{\mathsf{s}} N_2 : A$. We can therefore deduce $\Gamma \vdash_{\mathsf{s}} C[N_2] \equiv M_2 : B$ by replacing the derivation subtree of $\Gamma' \vdash_{\mathsf{s}} N_1 : A$ by the derivation tree of $\Gamma' \vdash_{\mathsf{s}} N_1 : A$ in the derivation tree of $\Gamma \vdash_{\mathsf{s}} C[N_1] : B$.

Otherwise N_1 is an almost safe application that is not safe and therefore N_1 is a strict subterm of M_1. In the derivation tree of a safe term, an almost safe application that is not safe can only occur as a premise of the abstraction rule. Thus the context $C[-]$ must be of the form $C'[\lambda\overline{y}.-]$ for some context $C'[-]$ and such that $\lambda\overline{y}.N_1$ is a safe term: $\Gamma'' \vdash_{\mathsf{s}} \lambda\overline{y}.N_1 : C$ for some Γ'', C. Applying the abstraction rule on N_2 gives $\Gamma'' \vdash_{\mathsf{s}} \lambda\overline{y}.N_2 : C$. Hence as in the previous case we can deduce $\Gamma \vdash_{\mathsf{s}} C[N_2] \equiv C'[\lambda\overline{y}.N_2] \equiv M_2 : B$ by substituting the derivation tree of $\Gamma'' \vdash_{\mathsf{s}} \lambda\overline{y}.N_2 : C$ for the derivation tree $\Gamma'' \vdash_{\mathsf{s}} \lambda\overline{y}.N_1 : C$ in the derivation tree of $\Gamma \vdash_{\mathsf{s}} M_1 : B$.

(ii) If N_1 is a safe term the we conclude as in (i). Otherwise, N_1 is an almost safe application: if $C[-] \equiv -$ then we can conclude immediately by Lemma 3.24(i); otherwise N_1 necessarily occurs as a subterm of a safe subterm of M_1 so we can conclude as in (i). $\square$

Remark 3.28 $\rightarrow_{\beta_s}$ *does not* preserve "unsafety": Take any safe annotated-term S and unsafe annotated-term U of the same type τ, then the term $(\lambda x^\tau y^\tau.y)\, U\, S : \tau$ is unsafe but it β_s-reduces to S which is safe.

3.1.3 Eta-long normal form

We now restrict our attention to the Church-style (safe) lambda calculus. Since terms are annotated, their type as well as the types of their subterms are uniquely determined. The η-expansion of $M : A \rightarrow B$ is defined as the annotated term $\lambda x^A.Mx : A \rightarrow B$ where $x : A$ is a fresh variable. The *η-long-expansion* of a term $M : (A_1,\ldots,A_n,o)$ is defined as $\lambda\varphi_1^{A_1}\ldots\varphi_l^{A_l}.M\varphi_1\ldots\varphi_l$ where each φ_i is a fresh variable. The *η-long normal form* (or just η-long nf) of an annotated term (also referred in the literature as *long reduced form*, *η-normal form* or *extensional form* [JP76, Hue75, Hue76]) is obtained by hereditarily η-expanding the body of every lambda abstraction as well as every subterm occurring in an *operand position* (*i.e.*, occurring as the second argument of some occurrence of the binary application operator). Formally,

Definition 3.29. The ***η-long normal form***, written $\lceil M \rceil$ or sometimes $\eta_{\mathsf{lnf}}(M)$, of an annotated term M of type $(A_1,\ldots,A_n,o)$ with $n \geq 0$ is defined by cases according to the syntactic shape of M (A simply-typed term is either an abstraction or it can be written uniquely as $s_0 s_1 \ldots s_m$ where $m \geq 0$ and s_0 is a variable, a Σ-constant or an abstraction.):

$$\begin{aligned}
\lceil \lambda x^\tau.N \rceil &\equiv \lambda x^\tau.\lceil N \rceil \\
\lceil \alpha N_1 \ldots N_m \rceil &\equiv \lambda\overline{\varphi}^{\overline{A}}.\alpha\lceil N_1 \rceil \ldots \lceil N_m \rceil \lceil \varphi_1 \rceil \ldots \lceil \varphi_n \rceil \\
\lceil (\lambda x^\tau.N) N_1 \ldots N_p \rceil &\equiv \lambda\overline{\varphi}^{\overline{A}}.(\lambda x^\tau.\lceil N \rceil)\lceil N_1 \rceil \ldots \lceil N_p \rceil \lceil \varphi_1 \rceil \ldots \lceil \varphi_n \rceil
\end{aligned}$$

where $m \geq 0$, $p \geq 1$, x is a variable, $\overline{\varphi} = \varphi_1 \ldots \varphi_n$ and each $\varphi_i : A_i$ is a fresh variable, and α is either a variable or a constant. The binder notation '$\lambda\overline{\varphi}^{\overline{A}}$' stands for '$\lambda\varphi_1^{A_1}\ldots\varphi_n^{A_n}$' if $n \geq 1$, and for 'λ' (called the *dummy lambda*) in the case $n = 0$.

Remark 3.30 The η-long normal form is defined for every simply-typed lambda-term, whether β-normal or not. Furthermore, the transformation does not introduce any new redex therefore the η-long normal form of a β-normal term is also β-normal.

Definition 3.31. We say that a safe annotated term is ***long-safe*** just if it is typable in the Church-like safe lambda calculus without using the rule $(\mathsf{app_{as}})$ from Def. 3.2. Equivalently, it is long-safe just if the judgement $\Gamma \vdash_{\mathsf{l}} M : T$ for some Γ, T can be derived from the system of rules of Table 3.2.

$$(\mathsf{var_l})\ \frac{}{\Gamma \vdash_{\mathsf{l}} x : A}\quad x : A \in \Gamma \qquad (\mathsf{wk_l})\ \frac{\Gamma \vdash_{\mathsf{l}} M : A}{\Delta \vdash_{\mathsf{l}} M : A}\quad \Gamma \subset \Delta$$

$$(\mathsf{app_l})\ \frac{\Gamma \vdash_{\mathsf{l}} M : (A_1, \ldots, A_n, B) \quad \Gamma \vdash_{\mathsf{l}} N_1 : A_1 \quad \ldots \quad \Gamma \vdash_{\mathsf{l}} N_n : A_n}{\Gamma \vdash_{\mathsf{l}} M N_1 \ldots N_n : B} \quad \operatorname{ord} \Gamma \geq \operatorname{ord} B$$

$$(\mathsf{abs_l})\ \frac{\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash_{\mathsf{l}} M : B}{\Gamma \vdash_{\mathsf{l}} \lambda x_1^{A_1} \ldots x_n^{A_n}.M : (A_1, \ldots, A_n, B)} \quad \operatorname{ord} \Gamma \geq \operatorname{ord} (A_1, \ldots, A_n, B)$$

Table 3.2: Typing rules for long-safe terms-in-contexts.

The terminology “long-safe” does not mean that those terms are in η-long normal form; the name is deliberately suggestive of a forthcoming lemma (Lemma 3.36). By definition, if an annotated term is long-safe then it is safe:

Lemma 3.32. $\Gamma \vdash_{\mathsf{l}} M : T \implies \Gamma \vdash_{\mathsf{s}} M : T$.

In general, long-safety is not preserved by η-expansion. For instance we have $\vdash_{\mathsf{l}} \lambda y^o z^o.y : (o,o,o)$ but performing one eta-expansion produces the term $\lambda x^o.(\lambda y^o z^o.y)x : (o,o,o)$ which is not long-safe. On the other hand, η-reduction (of one variable) preserves long-safety:

Lemma 3.33. $\Gamma \vdash_{\mathsf{l}} \lambda \varphi^\tau.M\,\varphi : A \ \wedge\ \varphi \notin FV(M) \implies \Gamma \vdash_{\mathsf{l}} M : A$.

Proof. Suppose $\Gamma \vdash_{\mathsf{l}} \lambda\varphi^\tau.M\,\varphi : A$. If M is an abstraction then by construction the annotated-term M is necessarily safe. If $M \equiv N_0 \ldots N_p$ with $p \geq 1$ then again, since $\lambda\varphi^\tau.N_0 \ldots N_p\varphi$ is safe, each of the N_i is safe for $0 \leq i \leq p$ and for every $z \in FV(\lambda\varphi^\tau.M\,\varphi)$, $\operatorname{ord} z \geq \operatorname{ord} \lambda\varphi^\tau.M\,\varphi = \operatorname{ord} M$. Since φ does not occur free in M we have $FV(M) = FV(\lambda\varphi^\tau.M\,\varphi)$, thus we can use the application rule to form $\Gamma_M \vdash_{\mathsf{l}} N_0 \ldots N_p : A$ where Γ_M is the subset of Γ satisfying $\operatorname{dom}(\Gamma) = FV(M)$. The weakening rules permits us to conclude $\Gamma \vdash_{\mathsf{l}} M : A$. ☐

Lemma 3.34 (Long-safety is preserved by η-long expansion)**.** $\Gamma \vdash_{\mathsf{l}} M : A \implies \Gamma \vdash_{\mathsf{l}} \lceil M \rceil : A$.

Proof. We first show that for every variable or constant $x : A$ we have $x : A \vdash_{\mathsf{l}} \lceil x \rceil : A$ by induction on $\operatorname{ord} x$. For ground type variable we have $x = \lceil x \rceil$ thus the property clearly holds. Step case: $A = (A_1, \ldots, A_n, o)$ with $n > 0$. Let $\varphi_i : A_i$ be a fresh variable for $1 \leq i \leq n$. Since $\operatorname{ord} A_i < \operatorname{ord} x$ the induction hypothesis gives $\varphi_i : A_i \vdash_{\mathsf{l}} \lceil \varphi_i \rceil : A_i$. Using $(\mathsf{wk_l})$ we obtain $x : A, \overline{\varphi} : \overline{A} \vdash_{\mathsf{l}} \lceil \varphi_i \rceil : A_i$. The application rule gives $x : A, \overline{\varphi} : \overline{A} \vdash_{\mathsf{l}} x\lceil\varphi_1\rceil \ldots \lceil\varphi_n\rceil : o$ and the abstraction rule gives $x : A \vdash_{\mathsf{l}} \lambda\overline{\varphi}.x\lceil\varphi_1\rceil \ldots \lceil\varphi_n\rceil \equiv \lceil x \rceil : A$.

We now prove the lemma by induction on M. The base case is covered by the previous observation. Step case:

- $M \equiv x N_1 \ldots N_m$ with $x : (B_1, \ldots, B_m, A)$, $A = (A_1, \ldots, A_n, o)$ for some $m \geq 0$, $n > 0$ and $N_i : B_i$ for $1 \leq i \leq m$. Let $\varphi_i : A_i$ be fresh variables for $1 \leq i \leq n$. By the previous observation we have $\varphi_i : A_i \vdash_{\mathsf{l}} \lceil\varphi_i\rceil : A_i$, the weakening rule then gives us $\Gamma, \overline{\varphi} : \overline{A} \vdash_{\mathsf{l}} \lceil\varphi_i\rceil : A_i$. Since the judgement $\Gamma \vdash_{\mathsf{l}} x N_1 \ldots N_m : A$ is formed using the $(\mathsf{app_l})$ rule, each N_j must be long-safe for $1 \leq j \leq m$, thus by the induction hypothesis we have $\Gamma \vdash_{\mathsf{l}} \lceil N_j \rceil : B_j$ and by weakening we get $\Gamma, \overline{\varphi} : \overline{A} \vdash_{\mathsf{l}} \lceil N_j \rceil : B_j$. The $(\mathsf{app_l})$ rule then gives $\Gamma, \overline{\varphi} : \overline{A} \vdash_{\mathsf{l}} x\lceil N_1 \rceil \ldots \lceil N_m \rceil \lceil \varphi_1 \rceil \ldots \lceil \varphi_n \rceil : o$. Finally the $(\mathsf{abs_l})$ rule gives $\Gamma \vdash_{\mathsf{l}} \lambda \overline{\varphi}.x\lceil N_1 \rceil \ldots \lceil N_m \rceil \lceil \varphi_1 \rceil \ldots \lceil \varphi_n \rceil \equiv \lceil M \rceil : A$, the side-condition of $(\mathsf{abs_l})$ being satisfied since $\operatorname{ord} \lceil M \rceil = \operatorname{ord} M$.

- $M \equiv N_0 \ldots N_m$ where $m \geq 1$ and N_0 is an abstraction. The the eta-long normal form of M is $\lceil M \rceil \equiv \lambda \overline{\varphi}.\lceil N_0 \rceil \ldots \lceil N_m \rceil \lceil \varphi_1 \rceil \ldots \lceil \varphi_n \rceil$ for some fresh variables $\varphi_1, \ldots, \varphi_n$. Again, using the induction hypothesis we can easily derive $\lceil M \rceil : A$.
- $M \equiv \lambda \overline{\eta}^{\overline{B}}.N$ where $A = (\overline{B}, C)$ and N is not an abstraction. The induction hypothesis gives $\Gamma, \overline{\eta} : \overline{B} \vdash_{\mathsf{l}} \lceil N \rceil : C$ and using $(\mathsf{abs_l})$ we get $\Gamma \vdash_{\mathsf{l}} \lambda \overline{\eta}.\lceil N \rceil \equiv \lceil M \rceil : A$.

□

Remark 3.35

1. The converse of this lemma does not hold: performing η-reduction over a large abstraction does not in general preserve long-safety. (This does not contradict Lemma 3.33 which states that safety is preserved when performing η-reduction on an abstraction of a *single* variable.) The simplest counter-example is the term $f^{(o,o,o)} \vdash_{\mathsf{st}} \lambda x^o.f\underline{x}$ which is not long-safe and whose eta-long normal form $f^{(o,o,o)} \vdash_{\mathsf{l}} \lambda x^o y^o.fxy$ is long-safe. Even for closed terms the converse does not hold: $\lambda f^{(o,o,o)} g^{((o,o,o),o)}.g(\lambda x^o.f\underline{x})$ is not long-safe but its eta-long normal form $\lambda f^{(o,o,o)} g^{((o,o,o),o)}.g(\lambda x^o y^o.fxy)$ is long-safe. In fact even the closed $\beta\eta$-normal term $\lambda f^{(o,(o,o),o,o)} g^{((o,o),o,o,o),o)}.g(\lambda y^{(o,o)} x^o.f\underline{x}y)$ which is not long-safe has a long-safe η-long normal form!

2. In an eta-long normal term, applications occurring in it can always be chosen large enough so that the side-condition of the rule $(\mathsf{app_l})$ is satisfied. Hence if a term is still not long-safe after η-long expansion, then it is necessarily due to some occurrence of an abstraction in the term for which the side-condition of the abstraction rule is not satisfied.

Lemma 3.36. *An annotated term $M \in \lambda_{\mathbb{T}}$ is safe if and only if its η-long normal form is long-safe; formally:*

$$\Gamma \vdash_{\mathsf{s}} M : T \iff \Gamma \vdash_{\mathsf{l}} \lceil M \rceil : T \ .$$

Proof. (Only if) Let $\Gamma \vdash_{\mathsf{s}} M : (A_1, \ldots, A_l, o)$. We show the result by induction on the structure of M. The base cases are trivial. Abstraction: M has the form $\lambda \overline{y}.M_0 \ldots M_p$ for some safe terms M_k, $0 \leq k \leq p$, $p \geq 0$. By the subject reduction lemma we have $\Gamma_M \vdash_{\mathsf{s}} M : (A_1, \ldots, A_l, o)$ where Γ_M is the subset of Γ containing only typing for free variables in M. The η-long expansion of M is $\lambda \overline{y} x_1..x_l.\lceil M \rceil \lceil x_1 \rceil \ldots \lceil x_l \rceil$ for some variables $x_1 : A_1, \ldots, x_l : A_l$ fresh in M. Let k range in $\{1..l\}$. By Lemma 3.34 and 3.32, each $\lceil x_k \rceil$ is safe, and by the I.H. $\lceil M \rceil$ is also safe. Therefore by $(\mathsf{app_{as}})$, $\lceil M \rceil \lceil x_1 \rceil \ldots \lceil x_l \rceil$ is an almost safe application. By Lemma 3.8, all the free variables of M have order greater than $\operatorname{ord}(A_1, \ldots, A_l, o)$, hence we can use the abstraction rule to form the judgement $\Gamma_M \vdash_{\mathsf{s}} \lambda \overline{y} x_1..x_l.\lceil M \rceil \lceil x_1 \rceil \ldots \lceil x_l \rceil : (A_1, \ldots, A_l, o)$ and the weakening rule permits us to conclude. The application case is treated identically.

(If) By induction on the structure of the Church-like term M: The variable and constant cases are trivial. Suppose that M is an application of the form $xN_1 \ldots N_m : A$ for $m \geq 1$. Its η-long normal form is $x\lceil N_1 \rceil \ldots \lceil N_m \rceil \lceil \varphi_1 \rceil \ldots \lceil \varphi_m \rceil : o$ for some fresh variables $\varphi_1, \ldots \varphi_m$. By assumption this term is long-safe therefore we have $\operatorname{ord} A \leq \operatorname{ord} \Gamma$ and for $1 \leq i \leq m$, $\lceil N_i \rceil$ is also long-safe. By the induction hypothesis this implies that each N_i is safe. We can then form the judgement $\Gamma \vdash_{\mathsf{s}} xN_1 \ldots N_m : A$ using the rules (var) and (app) (this is allowed since we have $\operatorname{ord} A \leq \operatorname{ord} \Gamma$). The case $M \equiv (\lambda x.N)N_1 \ldots N_m$ for $m \geq 1$ is treated identically.

Suppose that $M \equiv \lambda \overline{x}^{\overline{B}}.N : A$. By assumption, its η-long n.f. $\lambda \overline{x}^{\overline{B}} \overline{\varphi}^{\overline{C}}.\lceil N \rceil \lceil \varphi_1 \rceil \ldots \lceil \varphi_m \rceil : A$ (for some fresh variables $\overline{\varphi} = \varphi_1 \ldots \varphi_m$) is long-safe. Thus we have $\operatorname{ord} A \leq \operatorname{ord} \Gamma$. Furthermore the long-safe subterm $\lceil M \rceil \lceil \varphi_1 \rceil \ldots \lceil \varphi_m \rceil$ is precisely the eta-long normal form of $M\varphi_1 \ldots \varphi_m : o$ therefore by the induction hypothesis we have that $M\varphi_1 \ldots \varphi_m : o$ is safe. Since the φ_i's are all safe (by rule (var)), we can "peel-off" m applications of the rule $(\mathsf{app_{as}})$ (or (app)) from the sequent $\Gamma, \overline{x} : \overline{B}, \overline{\varphi} : \overline{C} \vdash_{\mathsf{s}} s\varphi_1 \ldots \varphi_m : o$ which gives us the sequent $\Gamma, \overline{x} : \overline{B}, \overline{\varphi} : \overline{C} \Vdash_{\mathsf{app}} M : A$. Since the variables $\overline{\varphi}$ are fresh for M, we can further peel-off one application of the weakening rule to obtain the judgement $\Gamma, \overline{x} : \overline{B} \vdash_{\mathsf{s}} M : A$. Finally we obtain $\Gamma \vdash_{\mathsf{s}} \lambda \overline{x}^{\overline{B}}.M : A$ using the rule (abs) (which is permitted since we have $\operatorname{ord} A \leq \operatorname{ord} \Gamma$). ☐

Proposition 3.37. *An annotated term $M \in \Lambda_{\mathbb{T}}$ is safe if and only if its η-long normal form is safe; formally:*

$$\Gamma \vdash_{\mathsf{s}} M : B \iff \Gamma \vdash_{\mathsf{s}} \lceil M \rceil : B \ .$$

Proof.

$$\begin{array}{llll}
\text{(If):} & \Gamma \vdash_{\mathsf{s}} \lceil M \rceil : T & \Longrightarrow \Gamma \vdash_{\mathsf{l}} \lceil M \rceil : T & \text{By Lemma 3.36 (only if),} \\
& & \Longrightarrow \Gamma \vdash_{\mathsf{s}} M : T & \text{By Lemma 3.36 (if).}
\end{array}$$

$$\begin{array}{llll}
\text{(Only if):} & \Gamma \vdash_{\mathsf{s}} M : T & \Longrightarrow \Gamma \vdash_{\mathsf{l}} \lceil M \rceil : T & \text{By Lemma 3.36 (only if),} \\
& & \Longrightarrow \Gamma \vdash_{\mathsf{s}} \lceil M \rceil : T & \text{By Lemma 3.32.}
\end{array}$$

☐

3.1.4 Almost safety

We now give an alternative presentation of the safe lambda calculus. Consider the Curry-style system of rules of Table 3.3. (The Church-style version of this system is obtained by annotating the λ-binder in the abstraction rule.)

$$(\mathsf{var_{as}})\ \frac{}{\Gamma \Vdash_{\mathsf{app}} x : A}\ x : A \in \Gamma \qquad (\mathsf{wk_{as}})\ \frac{\Gamma \Vdash_{\mathsf{app}} M : A}{\Delta \Vdash_{\mathsf{app}} M : A}\ \Gamma \subset \Delta \qquad (\mathsf{wk})\ \frac{\Gamma \vdash_{\mathsf{s}} M : A}{\Delta \vdash_{\mathsf{s}} M : A}\ \Gamma \subset \Delta$$

$$(\mathsf{app_{as}})\ \frac{\Gamma \Vdash_{\mathsf{app}} M : A \rightarrow B \quad \Gamma \vdash_{\mathsf{s}} N : A}{\Gamma \Vdash_{\mathsf{app}} M\, N : B} \qquad (\mathsf{abs_{as}})\ \frac{\Gamma, x : A \Vdash M : B}{\Gamma \Vdash \lambda x.M : A \rightarrow B}$$

$$(\delta)\ \frac{\Gamma \vdash_{\mathsf{s}} M : A}{\Gamma \Vdash_{\mathsf{app}} M : A} \qquad (\delta')\ \frac{\Gamma \Vdash_{\mathsf{app}} M : A}{\Gamma \Vdash M : A} \qquad (\rho)\ \frac{\Gamma \Vdash M : A}{\Gamma \vdash_{\mathsf{s}} M : A}\ \operatorname{ord} \Gamma \geq \operatorname{ord} A \ .$$

Table 3.3: Alternative definition of the safe lambda calculus *à la* Curry.

It is easy to see that these (Curry-style and Church-style) systems of rules are equivalent to the ones from Def. 3.2 in the sense that they generate the same set of judgements of the form $\Gamma \vdash_{\mathsf{s}} M : T$. The above systems, however, have the advantage of decomposing the application and abstraction rules into atomic steps where only one variable is abstracted at a time and only two terms are applied together at a time.

Definition 3.38. Terms typed with the entailment operator $\Vdash$ are called ***almost safe*** terms. Terms typed with the entailment operator $\Vdash_{\mathsf{app}}$ are called ***almost safe applications***.

The intuition behind these rules is that almost safe terms represent terms that are not safe but which can become safe if sufficiently many safe terms are applied to them or if sufficiently many variables are abstracted. The rule ($\mathsf{app_{as}}$) is used to form applications in which each applied term is safe:

Lemma 3.39.

1. *If $\Gamma \Vdash_{\mathsf{app}} M : T$ then $M \equiv N_0 \ldots N_m$ for some $m \geq 0$ where N_i is safe for every $0 \leq i \leq m$;*
2. *If $\Gamma \Vdash M : T$ then $M \equiv \lambda x_1 \ldots x_n.N_0 \ldots N_m$ for some $n, m \geq 0$ where N_i is safe for every $0 \leq i \leq m$.*

This result follows immediately from the definition of the rules.

The rule $(\mathsf{abs_{as}})$ is nothing less than the standard abstraction rule of the lambda calculus. As soon as the context and the type of the term being formed respect the safety condition (*i.e.*, all the context variables have order greater than the order of the type), the term can be marked as safe. This is done using the rule (ρ). Together with the rule (δ') this implies that the closure of an almost safe term is always safe:

Lemma 3.40. $\Gamma \Vdash_{\mathsf{app}} M : T \ \wedge \ \mathrm{dom}(\Gamma) = FV(M) \quad \Longrightarrow \quad \vdash_{\mathsf{s}} \mathsf{closure}(M) : T.$

The two weakening rules (wk) and $(\mathsf{wk_{as}})$ permit one to extend the context of a safe term or an almost safe application. We could have added a third rule to allow weakening for almost safe terms $\Gamma \Vdash M : T$ as well. This is however not necessary because this kind of weakening can always be eliminated. (In particular if the term is an abstraction then we can instead apply the rule $(\mathsf{wk_{as}})$ just before the abstraction rule).

An annotated term is almost safe if and only if its eta-long normal form is safe:

Lemma 3.41. *Let $M \in \Lambda_{\mathbb{T}}$. Then $\Gamma \Vdash M : T$ if and only if $\Gamma \Vdash \eta_{\mathsf{lnf}}(M) : T$.*

Proof. Only if: Let $\Gamma \Vdash M : T$ be an almost safe term. We proceed by induction on M. Suppose that the last rule used is (δ'). Then by Lemma 3.39 M is an application $N_0 N_1 \ldots N_k : (A_1, \ldots, A_n, B)$ with $k \geq 0$. Let φ_i for $i \in \{1..n\}$ be fresh variables, using the rules $(\mathsf{var_{as}})$, $(\mathsf{wk_{as}})$, $(\mathsf{app_{as}})$ and $(\mathsf{abs_{as}})$ we can build the almost safe term $\Gamma \Vdash \lambda \varphi_1^{A_1} \ldots \varphi_n^{A_n}.N_0 N_1 \ldots N_k \varphi_1 \ldots \varphi_n : T$.

If the last rule used is (δ) then M is safe therefore by Proposition 3.37, its eta-long normal form is safe and therefore by (δ) it is also almost safe. Otherwise the last rule used is $(\mathsf{abs_{as}})$, so by the induction hypothesis the eta-long nf of the premise is almost safe and we can conclude using $(\mathsf{abs_{as}})$.

If: It is again a proof by structural induction on the eta-long normal form. The basic idea is that we can "peel-off" applications of the rules $(\mathsf{abs_{as}})$ and $(\mathsf{app_{app}})$ introduced during the eta-expansion. ☐

The two preceding lemmas show that the closure of the eta-long normal form of an almost safe term is safe. This explains the expression "almost safe": an almost safe is semantically safe in the sense that it is (extensionally) equivalent to a safe term; on the other hand it is syntactically unsafe since it cannot appear as an operand of an application inside a larger safe term.

Lemma 3.42 (Safe beta reduction preserves almost safety)**.** *Let $M \rightarrow_{\beta_s} M'$. Then*

$$\Gamma \Vdash M : A \implies \Gamma \Vdash M' : A \ .$$

Proof. Suppose that $M \rightarrow_{\beta_s} M'$ and $\Gamma \Vdash M : A$. By Lemma 3.39, $M \equiv \lambda x_1 \ldots x_n.N_0 \ldots N_m$ for some $n, m \geq 0$ where N_i is safe for every $0 \leq i \leq m$. There are two cases: If the redex occurs in some N_i for $0 \leq i \leq m$ then we have $N \equiv \lambda x_1 \ldots x_n.N_0 \ldots N_i' \ldots N_m$ where $N_i \rightarrow_{\beta_s} N_i'$ for some N_i'. Since safety is preserved by safe reduction (Lemma 3.27), N_i' is safe. Hence we can conclude using the application and abstraction rule. The second case is when the redex is $N_1 \ldots N_q$ for some $1 \leq q \leq m$. This means that N_0 is of the form $\lambda y_1 \ldots y_q.P$ for some safe term P, and $M' \equiv P[N_1/y_1 \ldots N_q/y_q]N_{q+1} \ldots N_m$. The Substitution Lemma 3.19 and the application and abstraction rules permit us to conclude. ☐

3.1.5 Safety with respect to other type-ranking functions

We call ***type-ranking function*** any function $rank : \mathbb{T} \longrightarrow (L, \leq)$ mapping the set $\mathbb{T}$ of simple types over a set of atomic types $\mathbb{A}$ to some preorder $(L, \leq)$.

Example 3.43. The followings are examples of type-ranking functions $\mathbb{T} \longrightarrow (\mathbb{N}, \leq)$:

- Order: $\text{ord}(\alpha) = 0$ for $\alpha \in \mathbb{A}$, and $\text{ord}(A \to B) = \max(\text{ord}(A) + 1, \text{ord}(B))$;
- Height: $height(\alpha) = 0$ for $\alpha \in \mathbb{A}$ and $height(A \to B) = 1 + \max(height(A), height(B))$;
- Arity: $arity(\alpha) = 0$ for $\alpha \in \mathbb{A}$ and $arity(A \to B) = 1 + arity(B)$;
- Size: $size(\alpha) = 1$ for $\alpha \in \mathbb{A}$ and $size(A \to B) = 1 + size(A) + size(B)$;
- Number of atomic sub-types: $natoms(\alpha) = 1$ for $\alpha \in \mathbb{A}$ and $natoms(A \to B) = natoms(A) + natoms(B)$;
- Number of function space sub-types: $narrows(\alpha) = 0$ for $\alpha \in \mathbb{A}$ and $narrows(A \to B) = 1 + narrows(A) + narrows(B)$.

The pairing of two type-ranking functions is also a type-ranking function. For instance the pair $\langle \text{ord}, arity \rangle : \mathbb{T} \longrightarrow (\mathbb{N} \times \mathbb{N}, \leq)$ is a type-ranking function where $\leq$ denotes the lexicographic ordering.

We have defined the safe lambda calculus as a sub-language of the simply-typed lambda calculus obtained by restricting the occurrences of variables according to their *order*. *Does it make sense to define a version of the safe lambda calculus where the constraint relies on a different type-ranking function?*

In the safe lambda calculus, the application and abstraction rules permit us to perform multiple abstraction or application at a time. For the abstraction rule, the idea is that the side-condition might not be satisfied after one abstraction but it may become after consecutive abstractions, and similarly for the application rule. So by design, the typing system implicitly assumes that abstracting variables increases the order of the term's type, and inversely performing application decreases its order:

$$rank(A \to B) \geq rank(B) \ . \tag{3.1}$$

On the other hand, in order to prove the No-variable-capture Lemma we need the following property:

$$rank(A \to B) > rank(A) \ . \tag{3.2}$$

A type-ranking function verifying the above two conditions is said to be ***proper***.

Example 3.44. The type-ranking function *arity* is not proper. The type-ranking functions ord, *height*, *size*, *natoms* and *narrows* are all proper. Moreover ord is by definition the *minimal* proper function: any other proper function $rank : \mathbb{T} \longrightarrow (L, \leq)$ is greater than ord by pointwise ordering.

In turns out that this condition suffices to define a notion of safe lambda calculus verifying all the properties that we have proven up to now, including the No-variable-capture property. We can thus define a family of safe lambda calculi as follows: given a proper function *rank*, we define the *rank*-safe lambda calculus as the calculus obtained by replacing the references to the function ord by the function *rank* in the typing rules of Table 3.1. We say that a term is *rank*-safe if it is typable in the *rank*-safe lambda calculus.

Example 3.45. Consider the simply-typed term $g : \alpha \vdash_{\mathrm{Ch}} \lambda f^{\beta}.g : \beta \to \alpha$ for some types α, β. This term is *rank*-safe if and only if $rank(\alpha) > rank(\beta)$. Now take the types $\mu = o \to o \to o \to o$ and $\tau = (o \to o) \to o$. We have $\text{ord}\,\mu = 1 < 2 = \text{ord}\,\tau$ but $natoms(\mu) > 4 > 3 = natoms(\tau)$. Hence $g : \tau \vdash_{\mathrm{Ch}} \lambda f^{\mu}.g : \mu \to \tau$ is ord-safe but *natoms*-unsafe. Conversely, $g : \mu \vdash_{\mathrm{Ch}} \lambda f^{\tau}.g : \tau \to \mu$ is *natoms*-safe but ord-unsafe.

This example shows that there is no hierarchy of *rank*-safe lambda calculi: although we can compare two proper type-ranking functions, there is not necessarily an inclusion between the safe lambda calculi that they generate.

This thesis concerns only the ord-safe lambda calculus, but most of the presented results (except those pertaining to expressivity and complexity of the calculus) can easily be generalized to other proper ranking functions.

3.1.6 Maximality

We have presented the safe lambda calculus as a sub-language of the simply-typed lambda calculus verifying the so called No-variable-capture lemma (Lemma 3.15). A natural question is whether it is the *maximal* such language: *Is there a sub-language of the simply-typed lambda calculus containing the safe lambda calculus and verifying the No-variable-capture lemma?*

In fact not all sub-languages are of interest, we should only consider those verifying some basic properties. For instance we require the languages to have the *subterm property* (Lemma 3.10)—that every (large) subterm of a typable term is also tybable—which subsequently implies the subject reduction property (Lemma 3.27).

Let $\sqsupset$ denote a set of simply-typed terms-in-context. We write safe $\Lambda_{\rightarrow} + \sqsupset$ to denote the language consisting of the safe terms-in-context plus those of $\sqsupset$ (where the entailment operator $\vdash_{\mathsf{st}}$ is replaced by $\vdash_{\mathsf{s}}$) and closed by the typing rules of the safe lambda calculus.

Theorem 3.46. *Safe $\Lambda_{\rightarrow}$ is the maximal fragment of $\Lambda_{\rightarrow}$ containing the safe terms-in-context such that the subterm property and no-variable capture lemma hold.*

Proof. Let $\sqsupset$ be a set of simply-typed terms-in-context and consider the language Safe $\Lambda_{\rightarrow} + \sqsupset$. Suppose that it *strictly* contains the safe lambda calculus. Then $\sqsupset$ must contain an unsafe term-in-context. By definition, this term must have a large subterm of the form $M \equiv \lambda \overline{x}^{\overline{A}}.N$ where $\overline{x} = x_1, \ldots, x_k$, $\overline{A} = A_1, \ldots, A_k$, $k \geq 1$. for some term $N \equiv \cdots (\lambda y^B. \cdots x_i \cdots)$ that is not an abstraction for some $1 \leq i \leq k$ where $\operatorname{ord} B \geq \operatorname{ord} A_i$.

If Safe $\Lambda_{\rightarrow} + \sqsupset$ satisfies the subterm property then M must also be typable in Safe $\Lambda_{\rightarrow} + \sqsupset$: we have $\overline{z} : \overline{C} \vdash_{\mathsf{s}} M : (\overline{A}, D)$. But since we have $\operatorname{ord} B \geq \operatorname{ord} A_i$ we can also form the judgement $g : B \rightarrow A_i, y : B \vdash_{\mathsf{s}} g\,y : A$ for some fresh variables g, y not occurring free in M, thus by the rule ($\mathsf{app}_{\mathsf{as}}$) and ($\mathsf{abs}$) the term $\lambda \overline{z}^{\overline{C}} \overline{x}^{\overline{A}} g^{B \rightarrow A_i} y^B. M x_1 \ldots x_{i-1} (gy) x_{i+1} \ldots x_k$ belongs to Safe $\Lambda_{\rightarrow} + \sqsupset$. This term β_s-reduces to $\lambda \overline{z}^{\overline{C}} \overline{x}^{\overline{A}} g^{B \rightarrow A_i} y^B. N\,[gy/x_i]$. Now performing the substitution $N\,[gy/x_i] \equiv (\lambda y^B. \cdots x_i \cdots)\,[gy/x_i]$ without renaming variables would cause the capture of the variable y. Hence Safe $\Lambda_{\rightarrow} + \sqsupset$ does not verify the No-variable capture lemma. $\square$

Example 3.47. Consider the language Safe $\Lambda_{\rightarrow} + \{\vdash_{\mathsf{st}} M : (2, o)\}$ for $M \equiv \lambda f^2. f(\lambda x^o. f(\lambda y^o. x)$. It does not verify the subterm property: for instance the subterm $\lambda y^o.x$ is not typable. Further it does not verify the subject reduction lemma. Indeed it contains the term-in-context $f : 2 \vdash_{\mathsf{s}} M f : o$, but Mf reduces to $f(\lambda x^o. f(\lambda y^o. x))$ which is unsafe and different from M.

Now consider instead the language Safe $\Lambda_{\rightarrow} + \sqsupset$ where $\sqsupset$ contains the typing judgements for all M's subterms: $\sqsupset = \{\vdash_{\mathsf{st}} M : (2, o), f : 2 \vdash_{\mathsf{st}} f(\lambda x^o. f(\lambda y^o. x)) : 1, f : 2 \vdash_{\mathsf{st}} \lambda x^o. f(\lambda y^o. x) \equiv N : 1, f : 2, x : o \vdash_{\mathsf{st}} f(\lambda y^o. x) : o, x : o \vdash_{\mathsf{st}} \lambda y^o. x : 1\}$. In this language we can form the term $\lambda f^2 g^1 y^o. N(gy)$ (since $\operatorname{ord} y = 0 \geq \operatorname{ord}(gy) = 0$) which reduces to $\lambda f^2 g^1 y^o. f(\lambda y^o. x)\,[gy/x]$. Now the substitution cannot be performed without renaming variables as it would cause the capture of the variable y. Hence Safe $\Lambda_{\rightarrow} + \sqsupset$ does not verify the No-variable capture lemma.

3.1.7 Homogeneous safe lambda calculus

Our version of the safe lambda calculus does not make any assumption on types. In its original form however—in the setting of higher-order grammars—the safety restriction makes a further assumption on types called *homogeneity*. We recall from Sec. 2.2.2 that a type $(A_1, \ldots A_n, o)$ is said to be homogeneous whenever $\operatorname{ord} A_1 \geq \operatorname{ord} A_2 \geq \ldots \geq \operatorname{ord} A_n$ and each of the A_i is homogeneous. As defined in Sec. 3.2, the *homogeneous safe lambda calculus* denotes the restriction of the safe lambda calculus where types occurring in the derivation trees are all homogeneous. For the sake of completeness we now give a presentation of this calculus by means of a proper system of rules in which type homogeneity is implicitly enforced by the typing rules themselves.

We call ***stratified context*** any context of the form $x_{11} : A_{11}, \cdots, x_{1r} : A_{1r}, x_{21} : A_{21}, \ldots$ such that variables are listed in decreasing order and such that for every k, l and $i > j$, $\operatorname{ord} x_{ik} > \operatorname{ord} x_{jl}$. In other words, the context is stratified into lists of variables of the same orders, and the stratifications are arranged in strict decreasing order. Such stratified context will be abbreviated as

$$\overline{x_1} : \overline{A_1} \,|\, \cdots \,|\, \overline{x_n} : \overline{A_n} \ .$$

For every unstratified context Γ, we write $strat(\Gamma)$ to denote any possible valid stratification of Γ.

Definition 3.48. We define typing judgements of the form: $\overline{x_1} : \overline{A_1} \,|\, \cdots \,|\, \overline{x_n} : \overline{A_n} \vdash_{\mathsf{h}} M : B$ by induction over the following rules:

$$(\mathsf{h\text{-}const})\frac{}{\vdash_{\mathsf{h}} f : A}\ f : A \in \Sigma \qquad (\mathsf{h\text{-}var})\frac{}{\overline{x_1} : \overline{A_1} \,|\, \cdots \,|\, \overline{x_n} : \overline{A_n} \vdash_{\mathsf{h}} x_{ij} : A_{ij}} \qquad (\delta)\ \frac{\Gamma \vdash_{\mathsf{h}} M : A}{\Gamma \Vdash_{\mathsf{h.app}} M : A}$$

$$(\mathsf{h\text{-}wk})\frac{\Gamma \vdash_{\mathsf{h}} M : B \qquad \Gamma \subset \Delta}{\Delta \vdash_{\mathsf{h}} M : B} \qquad (\mathsf{perm})\frac{\Gamma \vdash_{\mathsf{h}} M : B \qquad \sigma(\Gamma)\ \text{homogeneous}}{\sigma(\Gamma) \vdash_{\mathsf{h}} M : B}$$

$$(\mathsf{h\text{-}app_{as}})\ \frac{\Gamma \vdash_{\mathsf{h}} s : (A_1, \ldots, A_n, B) \quad \Gamma \vdash_{\mathsf{h}} t_1 : A_1 \quad \ldots \quad \Gamma \vdash_{\mathsf{h}} t_n : A_n}{\Gamma \Vdash_{\mathsf{h.app}} s\ t_1 \ldots t_n : B}$$

$$(\mathsf{h\text{-}app_{strat}})\frac{\Gamma \vdash_{\mathsf{h}} N_0 : \big(B_{11}, \ldots, B_{1l} \,|\, \overline{B_2} \,|\, \cdots \,|\, \overline{B_m} \,|\, o\big) \quad \Gamma \vdash_{\mathsf{h}} N_1 : B_{11}\ \ldots\ \Gamma \vdash_{\mathsf{h}} N_l : B_{1l}}{\Gamma \vdash_{\mathsf{h}} N_0 N_1 \cdots N_l : (\overline{B_2} \,|\, \cdots \,|\, \overline{B_m} \,|\, o)}$$

$$(\mathsf{h\text{-}app_{partial}})\frac{\Gamma \vdash_{\mathsf{h}} M : \big(B_{11}, \ldots, B_{1l} \,|\, \overline{B_2} \,|\, \cdots \,|\, \overline{B_m} \,|\, o\big) \quad \Gamma \vdash_{\mathsf{h}} N : B_{11}}{\Gamma \vdash_{\mathsf{h}} MN : (B_{12}, \ldots, B_{1l} \,|\, \overline{B_2} \,|\ \cdots \,|\, \overline{B_m} \,|\, o)}\quad \operatorname{ord}\Gamma > \operatorname{ord} B_{11}$$

$$(\mathsf{h\text{-}abs})\frac{\overline{x_1} : \overline{A_1} \,|\cdots|\, \overline{x_{p+1}} : \overline{A_{p+1}} \,|\cdots|\overline{x_n} : \overline{A_n} \Vdash_{\mathsf{h.app}} M : B}{\overline{x_1} : \overline{A_1} \,|\, \cdots \,|\, \overline{x_p} : \overline{A_p} \vdash_{\mathsf{h}} \lambda \overline{x_{p+1}} \ldots \overline{x_n}.M : (\overline{A_{p+1}} \,|\ldots|\, \overline{A_n} \,|\, B)}\quad \operatorname{ord}\overline{A_n} \geq \operatorname{ord} B - 1$$

where Δ is an homogeneously-typed alphabet, Σ is a set of homogeneously-typed constants, and σ ranges over permutations on lists of type-assignments.

The main changes compared to the rules of the non-homogeneous safe lambda calculus are:

(i) The contexts are stratified;

(ii) All the types appearing in the rule are homogeneous;

(iii) The rule ($\mathsf{h\text{-}app_{as}}$) is the counterpart of rule ($\mathsf{app_{as}}$) in the safe lambda calculus: you can form an *homogeneous almost safe term* by applying several safe terms together;

(iv) The original application rule (app) is split into two rules: (a) ($\mathsf{h\text{-}app_{strat}}$) is a "stratified application". It applies an entire level of the type stratification. Because of type homogeneity, sufficiently many terms are applied to make the order of the term decrease, so no side-condition is necessary. (b) ($\mathsf{h\text{-}app_{partial}}$) is a partial application: it applies only two terms together provided that some condition on types is satisfied;

(v) Type-homogeneity constrains the order in which the variables are abstracted: in the rule ($\mathsf{h\text{-}abs}$), if a variable of a given order is abstracted then all the lower layers in the stratified context need to be abstracted as well;

(vi) Because of the previous point and because contexts are stratified, the side-condition present in the rule (abs) of the original safe lambda calculus is always satisfied and is hence not required here. Instead the side-condition in $(\mathsf{h\text{-}abs})$ ensures that the type $(\overline{A_n}|B)$ is homogeneous.

Lemma 3.49 (Basic properties). *Let $\Gamma \vdash_{\mathsf{h}} M : B$ be a valid judgement then*

(i) B is homogeneous;

(ii) $\forall z : A \in \Gamma : \; z \in FV(M) \implies \operatorname{ord} A \geq \operatorname{ord} B$;

(iii) (Context reduction) $\Gamma_M \vdash_{\mathsf{h}} M : B$ *where* $\Gamma_M = \{z : A \in \Gamma \mid z \in FV(M)\}$.

Proof. (i) and (ii) are proved by a trivial induction. (iii) Variables in Γ not occurring free in M are necessarily introduced by the weakening rule. The derivation of $\Gamma_M \vdash_{\mathsf{h}} M : A$ can thus be obtained by removing all the unnecessary applications of the weakening rule from the derivation tree of $\Gamma \vdash_{\mathsf{h}} M : A$. ☐

Proposition 3.50. *The judgement $strat(\Gamma) \vdash_{\mathsf{h}} M : T$ (resp. $strat(\Gamma) \Vdash_{\mathsf{h.app}} M : T$) is valid if and only if there is a derivation tree for $\Gamma \vdash_{\mathsf{s}} M : T$ (resp. $\Gamma \Vdash_{\mathsf{app}} M : T$) in the Curry-style safe lambda calculus (Def. 3.2) such that all the types appearing in the derivation tree are homogeneously-typed.*

Proof. Only if: The proof is by a trivial structural induction on $\Gamma \vdash_{\mathsf{h}} M : T$. *If*: We proceed by structural induction on the derivation tree of $\Gamma \vdash_{\mathsf{s}} M : T$. The cases (var), (const), (wk) and $(\mathsf{app_{as}})$ are trivial. Suppose that the rule (app) is used. Then we can form the equivalent homogeneous term by using the I.H. and applying $(\mathsf{app_{strat}})$ several times followed by one application of $(\mathsf{app_{partial}})$.

Abstraction: The sequent is of the form $\Gamma \vdash_{\mathsf{s}} \lambda x_1 \ldots x_n.s : (A_1, \ldots, A_n, B)$ with $\operatorname{ord}\Gamma \geq \operatorname{ord}(A_1, \ldots, A_n, B)$. By the induction hypothesis we have $strat(\Gamma, x_1 : A_1, \ldots, x_n : A_n) \Vdash_{\mathsf{h.app}} s : B$. Since we have $\operatorname{ord}\Gamma \geq \operatorname{ord}(A_1, \ldots, A_n, B)$, all the variables in Γ have order strictly greater than the variables $x_1, \ldots, x_n$. Therefore there exists a stratification of $\Gamma, x_1, \ldots, x_n$ of the form

$$strat(\Gamma) \,|\, \overline{y_1} : \overline{Y_1} \,|\, \cdots \,|\, \overline{y_l} : \overline{Y_l}$$

for some $l \geq 1$ such that the sequence of variables $\overline{y_1}, \ldots, \overline{y_l}$ is equal to $x_1, \ldots, x_n$. Hence using the permutation rule (perm) we can form the judgement

$$strat(\Gamma) \,|\, \overline{y_1} : \overline{Y_1} \,|\, \cdots \,|\, \overline{y_l} : \overline{Y_l} \Vdash_{\mathsf{h.app}} s : B \ .$$

We can now apply the rule $(\mathsf{h\text{-}abs})$ to form $strat(\Gamma) \Vdash_{\mathsf{h.app}} \lambda x_1 \ldots x_n.s : (A_1, \ldots, A_n, B)$. The side-condition of the rule is satisfied because $(A_1, \ldots, A_n, B)$ is homogeneous by assumption. ☐

Example 3.51.

(i) The untyped term $(\lambda f x.x) g y$ is homogeneously safe. One possible derivation is:

$$
(\mathsf{app_{strat}})\,\dfrac{
(\mathsf{wk})\,\dfrac{
(\mathsf{app_{strat}})\,\dfrac{
(\mathsf{wk})\,\dfrac{
(\mathsf{abs})\,\dfrac{
(\mathsf{wk})\,\dfrac{
(\mathsf{abs})\,\dfrac{
(\delta)\,\dfrac{
(\mathsf{var})\,\dfrac{}{x : o \vdash_{\mathsf{h}} x : o}
}{\Vdash_{\mathsf{h.app}} x : o}
}{\vdash_{\mathsf{h}} \lambda x.x : 1}
}{f : (o,o) \vdash_{\mathsf{h}} \lambda x.x : 1}
}{\vdash_{\mathsf{h}} \lambda f x.x : (1,o,o)}
}{g : (o,o) \vdash_{\mathsf{h}} \lambda f x.x : (1,o,o)}
\qquad
\dfrac{}{g : 1 \vdash_{\mathsf{h}} g : 1}\,(\mathsf{var})
}{g : 1 \vdash_{\mathsf{h}} (\lambda f x.x) g : 1}
}{g : 1, y : o \vdash_{\mathsf{h}} (\lambda f x.x) g : 1}
\qquad
\dfrac{
\dfrac{}{y : o \vdash_{\mathsf{h}} y : o}\,(\mathsf{var})
}{g : 1, y : o \vdash_{\mathsf{h}} y : o}\,(\mathsf{wk})
}{g : 1, y : o \vdash_{\mathsf{h}} (\lambda f x.x) g y : o}
$$

(ii) The annotated-terms $\lambda g^{(o,(o,o),o)} x^o.gx$ and $\lambda g^{(o,(o,o),o)} x^o.gx(\lambda x.x)$ are both safe but not homogeneously safe because they are not homogeneously typed. This shows that the safe lambda calculus strictly contains the homogeneous safe lambda calculus.

(iii) The annotated-term $\lambda x^0 f^1 \varphi^2.\varphi$ is safe but not homogeneously safe because its type $(0, 1, 2, 2)$ is not homogeneous. On the other hand, the untyped term $\lambda x f \varphi.\varphi$ is homogeneously safe because the annotation $\lambda x^0 f^0 \varphi^0.\varphi$ is safe and homogeneously typed.

Example 3.52. Take the following term:

$$E \equiv (\lambda a.a(\lambda b.a(\lambda cd.d)))(\lambda e.e(\lambda f.f)) \ .$$

(It was used by Sereni [Ser05] as a counter-example to show that not all simply-typed terms are *size-change terminating* [LJBA01].) The untyped term E is *universally safe*. Indeed, let $E' \in \Lambda_{\mathbb{T}}$ be a type-annotation of E (*i.e.*, $|E'| = E$) such that E' is typable in the Church simply-typed lambda calculus. Then it is easy to check that we have

$$\vdash_{\mathrm{Ch}} E' : A \rightarrow A$$

for some type $A \in \mathbb{T}$ (and thus E has for principal type $\alpha \rightarrow \alpha$) and the type assignments for the bound variables in E' are of the form:

$$\begin{aligned}
&a : C \rightarrow A \rightarrow A \\
&b : B \rightarrow B \\
&c : B \rightarrow B \\
&d : A \\
&e : C \equiv (B \rightarrow B) \rightarrow A \rightarrow A \\
&f : B
\end{aligned}$$

for some for some types $A, B \in \mathbb{T}$ (not necessarily atomic). It is then an easy exercise to check that for every type $A, B \in \mathbb{T}$, we can form the following term-in-context:

$$\vdash_{\mathsf{s}} E' : A \rightarrow A \ .$$

On the other hand, E is only homogeneously safe (and not universally homogeneously safe). More precisely, its annotation E' is *homogeneously* safe if and only if $\operatorname{ord} B \geq \operatorname{ord} A - 1$. Formally:

$$\vdash_{\mathsf{h}} E' : A \rightarrow A \qquad \Longleftrightarrow \qquad \operatorname{ord} B \geq \operatorname{ord} A - 1 \ .$$

(In particular, the condition in the right-hand side implies that A, B and the types of a, b, c, d, e, f are all homogeneous.)

Remark 3.53 (Related work) In her thesis, de Miranda proposed a different notion of safe lambda calculus [dM06]. This notion corresponds to (a less general version of) our notion of *homogeneous* safe lambda calculus: the applicative fragment (*i.e.*, without lambda-abstraction) of de Miranda's typing system coincides with the applicative fragment of the system of Def. 3.48. In particular a version of Proposition 3.11 is shown by de Miranda [dM06]. In the presence of lambda abstraction, however, our system is less restrictive. For instance the judgement $\vdash_{\mathsf{h}} \lambda f^{(o,o,o)} x^o.fx : (o, o)$ is derivable in the homogeneous safe lambda calculus but not in the safe lambda calculus *à la* de Miranda. One can show that the system introduced by de Miranda is in fact equivalent to the fragment of the *long-safe lambda calculus* (Def. 3.31) restricted to homogeneous types.

3.2 Complexity

This section is concerned with the complexity of the beta-eta equivalence problem for the safe lambda calculus: *Given two safe lambda-terms, are they equivalent up to $\beta\eta$-conversion?*

3.2.1 Statman's result

Let $\exp_h(m)$ denote the *tower of exponential* function defined by:

$$\exp_0(m) = m$$
$$\exp_{h+1}(m) = 2^{\exp_h(m)} \ .$$

A program is ***elementary recursive*** if its run-time can be bounded by $\exp_K(n)$ for some constant K where n is the length of the input. We recall the definition of finite type theory. We define $\mathcal{D}_0 = \{\mathbf{true}, \mathbf{false}\}$ and $\mathcal{D}_{k+1} = \mathscr{P}(\mathcal{D}_k)$ (*i.e.*, the powerset of $\mathcal{D}_k$). For $k \geq 0$, we write x^k, y^k and z^k to denote variables ranging over $\mathcal{D}_k$. Prime formulae are x^0, $\mathbf{true} \in y^1$, $\mathbf{false} \in y^1$, and $x^k \in y^{k+1}$. Formulae are built up from prime formulae using the logical connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$ and the quantifiers $\forall$ and $\exists$. Meyer showed that deciding the validity of such formulae requires nonelementary time [Mey74].

A famous result by Statman states that deciding the $\beta\eta$-equality of two first-order typable lambda-terms is not elementary recursive [Sta79b]. The proof proceeds by encoding the Henkin quantifier elimination of type theory in the simply-typed lambda calculus and by appealing to Meyer's result [Mey74]. Simpler proofs have subsequently been given: one by Mairson [Mai92] and another by Loader [Loa98a]. Both proceed by encoding the Henkin quantifier elimination procedure in the lambda calculus, as in the original proof, but their use of list iteration to implement quantifier elimination makes them much easier to understand.

It turns out that all these encodings rely on unsafe terms: Statman's encoding uses the conditional function `sg` which is not definable in the safe lambda calculus [BO07]; Mairson's encoding uses unsafe terms to encode both quantifier elimination and set membership, and Loader's encoding uses unsafe terms to build list iterators. We are thus led to conjecture that finite type theory (see definition in Sec. 3.2.2) is intrinsically unsafe in the sense that every encoding of it in the lambda calculus is necessarily unsafe. Of course this conjecture does not rule out the possibility that another non-elementary problem is encodable in the safe lambda calculus.

3.2.2 Mairson's encoding

We refer the reader to Mairson's original paper [Mai92] for a detailed account of his encoding. We show here why Mairson's encoding does not work in the safe lambda calculus. We then introduce a variation that eliminates some of the unsafety. Although the resulting encoding does not suffice to interpret type theory in the safe lambda calculus, it enables another interesting encoding: that of the True Quantifier Boolean Formula (TQBF) problem. This implies that deciding beta-eta equality of safe terms is PSPACE-hard.

3.2.2.1 Sources of unsafety

In Mairson's encoding, boolean values are encoded by terms of type $\mathsf{B} = \sigma \rightarrow \sigma \rightarrow \sigma$ for some type σ, and variables of order $k \geq 0$ are encoded by terms of type Δ_k defined as $\Delta_0 \equiv \mathsf{B}$ and $\Delta_{k+1} \equiv (\Delta_k \rightarrow \tau \rightarrow \tau) \rightarrow \tau \rightarrow \tau$ for any type τ. Using this encoding, unsafety manifests itself in three different places:

(i) *Set membership:* The prime formula "$x^k \in y^{k+1}$" is encoded as

$$x : \Delta_k, y : \Delta_{k+1} \vdash_{\mathsf{st}} y(\lambda z^{\Delta_k}.OR(eq_k\ \underline{x}\ z)\ F : \Delta_k \rightarrow \Delta_{k+1} \rightarrow \Delta_0 \qquad (3.3)$$

for some terms OR, F, eq_k. This term is unsafe because of the underline occurrence of x^k which is not abstracted together with y^k.

(ii) *Quantifier elimination* is implemented using a list iterator $\mathbf{D}_{k+1}$ of type Δ_{k+2} which acts like the `foldr` function (from functional programming) over the list of all elements of $\mathcal{D}_k$.

Thus nested quantifiers in the formula are encoded by nested list iterations. This can be source of unsafety, for instance the formula "$\forall x^0.\exists y^0.x^0 \vee y^0$" is encoded as

$$\vdash_{\mathsf{st}} \mathbf{D}_0(\lambda x^{\Delta_0}.AND(\mathbf{D}_0(\lambda y^{\Delta_0}.OR(\underline{x} \vee y))F))\ T : \mathsf{B}$$

for some terms AND, OR, F and T and where the type τ is instantiated as B. This term is unsafe due to the underlined occurrence which is unsafely bound.

More generally, nested binding will be encoded safely if and only if every variable x in the formula is bound by the first quantifier $\exists z$ or $\forall z$ satisfying $\operatorname{ord} z \geq \operatorname{ord} x$ in the path to the root of the formula AST. So for example if set-membership were safely encodable then the interpretation of "$\forall x^k.\exists y^{k+1}.x^k \in y^{k+1}$" would be unsafe whereas that of "$\forall y^{k+1}.\exists x^k.x^k \in y^{k+1}$" would be safe.

(iii) *Elements of the type hierarchy.* The base set $\mathcal{D}_0$ of booleans is represented by a safe term $\mathbf{D}_0$ of type Δ_0. Higher-order sets $\mathcal{D}_k$ for $k \geq 1$ are represented by unsafe terms $\mathbf{D}_k$: they are constructed from $\mathbf{D}_0$ using a powerset construction that is unsafe.

The second source of unsafety can be easily overcome, the idea is as follows. We introduce multiple domains of representation for a given formula. An element of $\mathcal{D}_k$ is thereby represented by countably many terms of type Δ_k^n where $n \in \mathbb{N}$ indicates the level of the domain of representation. The type Δ_k^n is defined in such a way that its order strictly increases as n grows. Furthermore, there exists a term that can lower the domain of representation of a given term. Thus each formula variable can have a different domain of representation, and since there are infinitely many such domains, it is always possible to find an assignment of representation domains to variables such that the resulting encoding term is safe.

There is no obvious way to eliminate unsafety in the two other cases however. For instance in the case of set-membership, Mairson's encoding (3.3) could be made safe by appealing to a term that changes the domain of representation of an encoded higher-order value of the type-hierarchy. Unfortunately, such transformation is intrinsically unsafe! (Lowering the level of representation is safe, increasing it is unsafe.)

In the following paragraphs we present in detail a variation over Mairson's encoding in which quantifier elimination is safely encoded.

3.2.2.2 Encoding basic boolean operations

Let o be a base type and define the family of types $\sigma_0 \equiv o$, $\sigma_{n+1} \equiv \sigma_n \rightarrow \sigma_n$ satisfying $\operatorname{ord} \sigma_n = n$. Booleans are encoded over domains $\mathsf{B}_n \equiv \sigma_n \rightarrow o \rightarrow o \rightarrow o$ for $n \geq 0$, each type B_n being of order $n+1$. We write $\underline{i}_{n+1}$ to denote the term $\lambda x^{\sigma_n}.x$ of type σ_{n+1} for $n \geq 0$. The truth values **true** and **false** are represented by the following closed terms parameterized by $n \in \mathbb{N}$:

$$\begin{aligned} T^n &\equiv \lambda u^{\sigma_n} x^o y^o.x : \mathsf{B}_n \\ F^n &\equiv \lambda u^{\sigma_n} x^o y^o.y : \mathsf{B}_n \ . \end{aligned}$$

Clearly these terms are safe. Moreover the following relations hold for all $n, n' \geq 0$:

$$\begin{aligned} \lambda u^{\sigma_{n'}}.T^{n+1}\ \underline{i}_{n+1} &\rightarrow_\beta T^{n'} \\ \lambda u^{\sigma_{n'}}.F^{n+1}\ \underline{i}_{n+1} &\rightarrow_\beta F^{n'} \ . \end{aligned}$$

It is then possible to change the domain of representation of a Boolean value from a higher-level to another arbitrary level using the conversion term:

$$\mathbf{C}_0^{n+1 \mapsto n'} \equiv \lambda m^{\mathsf{B}_{n+1}} u^{\sigma_{n'}}.m\ \underline{i}_{n+1} : \mathsf{B}_{n+1} \rightarrow \mathsf{B}_{n'}$$

so that if a term M of type B_n, for $n \geq 1$, is beta-eta convertible to T^n (resp. F^n) then $\mathbf{C}_0^{n \mapsto n'}\ M$ of type $\mathsf{B}_{n'}$ is beta-eta convertible to $T^{n'}$ (resp. $F^{n'}$).

Observe that although $\mathbf{C}_0^{n+1 \mapsto n'}$ is safe for all $n, n' \geq 0$, if we apply a variable to it then the resulting term-in-context

$$x : B_{n+1} \vdash_{\mathsf{st}} \mathbf{C}_0^{n+1 \mapsto n'}\ x : B_n$$

is safe if and only if $\operatorname{ord} B_{n+1} \geq \operatorname{ord} B_{n'}$, that is to say if and only if the transformation decreases the domain of representation of x.

Boolean functions are encoded by the following closed safe terms parameterized by n:

$$\begin{aligned}
AND^n &\equiv \lambda p^{\mathsf{B}_n} q^{\mathsf{B}_n} u^{\sigma_n} x^o y^o . p\ u\ (q\ u\ x\ y)\ y : \mathsf{B}_n \to \mathsf{B}_n \to \mathsf{B}_n \\
OR^n &\equiv \lambda p^{\mathsf{B}_n} q^{\mathsf{B}_n} u^{\sigma_n} x^o y^o . p\ u\ x\ (q\ u\ x\ y) : \mathsf{B}_n \to \mathsf{B}_n \to \mathsf{B}_n \\
NOT^n &\equiv \lambda p^{\mathsf{B}_n} u^{\sigma_n} x^o \lambda y^o . p\ u\ y\ x : \mathsf{B}_n \to \mathsf{B}_n \to \mathsf{B}_n\ .
\end{aligned}$$

3.2.2.3 Coding elements of the type hierarchy

For every $n \in \mathbb{N}$ we define the hierarchy of type Δ_k^n as follows: $\Delta_0^n \equiv \mathsf{B}_n$ and $\Delta_{k+1}^n \equiv \Delta_k^{n*}$ where for a given type α, $\alpha^* = (\alpha \to \tau \to \tau) \to \tau \to \tau$ for *any* type τ. We encode an occurrence x^k of a *formula variable* by a *term variable* x^k of type Δ_k^n for some level of domain representation $n \in \mathbb{N}$. Following Mairson's encoding, each set $\mathcal{D}_k$ is represented by a term $\mathbf{D}_k^n$ encoding the list of all its elements:

$$\begin{aligned}
\mathbf{D}_0^n &\equiv \lambda c^{\mathsf{B}_n \to \tau \to \tau} e^\tau . c\ T^n\ (c\ F^n\ e) : \Delta_1^n \\
\mathbf{D}_{k+1}^n &\equiv powerset_{\Delta_k^n}\ \mathbf{D}_k^n : \Delta_{k+2}^n
\end{aligned}$$

where

$$\begin{aligned}
powerset_\alpha \equiv\ & \lambda A^{*(\alpha \to \alpha^{**} \to \alpha^{**}) \to \alpha^{**} \to \alpha^{**}} . \\
& \quad A^*\ double_\alpha\ (\lambda c^{\alpha^* \to \tau \to \tau} b^\tau . c\ (\lambda c'^{\alpha \to \tau \to \tau} b'^\tau . b')\ b) \\
& : ((\alpha \to \alpha^{**} \to \alpha^{**}) \to \alpha^{**} \to \alpha^{**}) \to \alpha^{**} \\
double_\alpha \equiv\ & \lambda x^\alpha\ l^{(\alpha^* \to \tau \to \tau) \to \tau \to \tau}\ c^{\alpha^* \to \tau \to \tau}\ b^\tau . \\
& \quad l (\lambda e^{\alpha^*} . c\ (\lambda c'^{\alpha \to \tau \to \tau}\ b'^\tau . c'\ \underline{x}\ (e\ c'\ b')))(l\ c\ b) \\
& : \alpha \to \alpha^{**} \to \alpha^{**}\ .
\end{aligned}$$

(In the definition of $\mathbf{D}_{k+1}^n$, to see why it is possible to apply $powerset_{\Delta_k^n}$ and $\mathbf{D}_k^n$ one needs to understand that the term $\mathbf{D}_k^n$ is of type Δ_{k+1}^n *polymorphic in* τ. The application can thus be typed by taking $\tau \equiv \Delta_{k+2}^n$ in the term $\mathbf{D}_k^n$.)

Observe that the term *double* is unsafe because the underlined variable occurrence x is not bound together with c'. Consequently for all $n \geq 0$, $\mathbf{D}_0^n$ is safe and $\mathbf{D}_k^n$ is unsafe for all $k > 0$.

3.2.2.4 Quantifier elimination

Terms of type Δ_{k+1}^n are now used as iterators over lists of elements of type Δ_k^n and we set $\tau \equiv \mathsf{B}_n$ in the type Δ_{k+1}^n in order to iterate a level-n Boolean function. Since $\operatorname{ord} \Delta_k^n \geq \operatorname{ord} \mathsf{B}_n$ for all n, all the instantiations of the terms $\mathbf{D}_k^n$ will be safe (although the terms $\mathbf{D}_k^n$ themselves are not safe for $k > 1$). Following [Mai92], quantifier elimination interprets the formula $\forall x^k . \Phi(x^k)$ as the iterated conjunction

$$\mathbf{C}_0^{n \mapsto 0} \left(\mathbf{D}_k^n (\lambda x^{\Delta_k^n} . AND^n (\hat{\Phi}\, x))\, T^n \right)\ ,$$

where $\hat{\Phi}$ is the interpretation of Φ and n is the representation level chosen for the variable x^k. Similarly we interpret $\exists x^k . \Phi(x^k)$ by the disjunction $\mathbf{C}_0^{n \mapsto 0} \left(\mathbf{D}_k^n (\lambda x^{\Delta_k^n} . AND^n (\hat{\Phi}\, x))\, T^n \right)$.

3.2.2.5 Encoding the formula

Given a formula of type theory, it is possible to encode it in the lambda calculus by inductively applying on the formula the above encodings of boolean operations and quantifiers, each variable occurrence in the formula being assigned some domain of representation.

We now show that there exists an assignment of representation domains for each variable occurrence such that the resulting term is safe. Let $x_p^{k_p} \ldots x_1^{k_1}$ for $p \geq 1$ be the list of variables appearing in the formula, given in order of appearance of their binder in the formula (*i.e.*, $x_p^{k_p}$ is bound by the leftmost binder). We fix the domain of representation of each variable as follows. The right-most variable $x_1^{k_1}$ is encoded in the domain $\Delta^0_{k_1}$; and if for $1 \leq i < p$ the domain of representation of $x_i^{k_i}$ is $\Delta^l_{k_l}$ then the domain of representation of $x_{i+1}^{k_{i+1}}$ is defined as $\Delta^{l'}_{k_{i+1}}$ where l' is the smallest natural number such that $\operatorname{ord} \Delta^{l'}_{k_{i+1}}$ is strictly greater than $\operatorname{ord} \Delta^{l}_{k_i}$.

This way, since variables that are bound first have higher order, variables that are bound in nested list-iterations—corresponding to nested quantifiers in the formula—are guaranteed to be safely bound.

Example 3.54. The formula $\forall x^0.\exists y^0.x^0 \vee y^0$, which is encoded by an unsafe term in Mairson's encoding, is represented in our encoding by the safe term

$$\vdash_{\mathsf{s}} \mathbf{C}_0^{1\mapsto 0} \Big(\mathbf{D}_0^1\ (\lambda x^{\Delta_0^1}.AND^0(\mathbf{D}_0^0\ (\lambda y^{\Delta_0^0}.OR^0(OR^0\ (\mathbf{C}_0^{1\mapsto 0}\ x)\ y))\ F^0))\ T^1\Big) : \mathsf{B}_0\ .$$

3.2.2.6 Set-membership

To complete the interpretation of prime formulae, we need to show how to encode set membership. Unfortunately, the introduction of multiple domains of representation does not suffice to completely eliminate the unsafety of Mairson's encoding of set membership.

Indeed, adapting Mairson's encoding of set membership requires the ability to perform conversion of domains of representation for higher-order sets (not only for Boolean values). The conversion term $\mathbf{C}_0^{n+1\mapsto n'}$ can be generalized to higher-order sets as follows:

$$\mathbf{C}_{k+1}^{n\mapsto n'} \equiv \lambda m^{\Delta^n_{k+1}} u^{\Delta^n_k \to \tau \to \tau} v^\tau . m(\lambda z^{\Delta^n_k} w^\tau . \underline{u(\underline{\mathbf{C}_k^{n\mapsto n'} z})w})v : \Delta^n_{k+1} \to \Delta^{n'}_{k+1}$$

where $k \geq 0$. Unfortunately this term is safe if and only if $n = n'$ (The largest underlined subterm is safe just when $n \geq n'$ and the other underline subterm is safe just when $n' \geq n$). Hence at higher-orders, all the non-trivial conversion terms are unsafe.

If the terms $\mathbf{C}_{k+1}^{n\mapsto n'}$, $k \geq 0$, $n \neq n'$ were safely representable then the encoding would go as follows: We set $\tau \equiv \mathsf{B}_0$ in the types Δ^n_{k+1} for all $n, k \geq 0$ in order to iterate a level-0 Boolean function. Firstly, the formulae "**true** $\in y^1$" and "**false** $\in y^1$" can be encoded by the safe terms $y^1(\lambda x^0.OR^0\ x^0)F^0$ and $y^1(\lambda x^0.OR^0(NOT^0\ x^0))F^0$ respectively. For the general case "$x^k \in y^{k+1}$" we proceed as in Mairson's proof [Mai92]: we introduce lambda-terms encoding set equality, set membership and subset tests, and we further parameterize these encodings by a natural number n.

$$\begin{aligned}
member^{n+1}_{k+1} &\equiv \lambda x^{\Delta^{n+1}_k} y^{\Delta^{n+1}_{k+1}}.(\mathbf{C}_{k+1}^{n+1\mapsto n}\ y)\ (\lambda z^{\Delta^n_k}.OR^0(eq^n_k\ (\mathbf{C}_k^{n+1\mapsto n}\ x)\ z))\ F^0 \\
&: \Delta^{n+1}_k \to \Delta^{n+1}_{k+1} \to \mathsf{B}_0 \\
subset^n_{k+1} &\equiv \lambda x^{\Delta^n_{k+1}} y^{\Delta^n_{k+1}}.x\ (\lambda x^{\Delta^n_k}.AND^0(member^n_{k+1}\ x\ y))\ T^0 \\
&: \Delta^n_{k+1} \to \Delta^n_{k+1} \to \mathsf{B}_0 \\
eq^n_0 &\equiv \lambda x^{\mathsf{B}_n}.\lambda y^{\mathsf{B}_n}.\mathbf{C}_0^{n\mapsto 0}\ \ (OR^n(AND^n\ x\ y)(AND^n(NOT^n\ x)(NOT^n\ y))) \\
&: \mathsf{B}_n \to \mathsf{B}_n \to \mathsf{B}_0 \\
eq^n_{k+1} &\equiv \lambda x^{\Delta^n_{k+1}}\ y^{\Delta^n_{k+1}}.(\lambda op^{\Delta^n_{k+1}\to\Delta^n_{k+1}\to\mathsf{B}_0}.AND^0(op\ x\ y)(op\ y\ x))\ subset^n_{k+1} \\
&: \Delta^n_{k+1} \to \Delta^n_{k+1} \to \mathsf{B}_0\ .
\end{aligned}$$

The variables in the definition of eq^n_{k+1} and $subset^n_{k+1}$ are safely bounds. Moreover, the occurrence of x in $member^{n+1}_{k+1}$ is now safely bound—which was not the case in Mairson's original encoding—thanks to the fact that the representation domain of z is lower than that of x. The formula $x^k \in y^{k+1}$ can then be encoded as

$$x : \Delta^n_k, y : \Delta^{n'}_{k+1} \vdash_{\mathsf{st}} member^u_{k+1}\ (\mathbf{C}^{n\mapsto u}_k\ x)\ (\mathbf{C}^{n'\mapsto u}_{k+1}\ y) : \mathsf{B}_0$$

for some $n, n' \geq 2$ and $u = \min(n, n') + 1$.

Unfortunately this encoding is not completely safe because, as mentioned before, the conversion term $\mathbf{C}^{n\mapsto u}_k$ is unsafe for $k \geq 1$, $n \neq u$. We conjecture that the set-membership function is intrinsically unsafe.

3.2.3 PSPACE-hardness

We observe that instances of the True Quantified Boolean Formulae satisfaction problem (TQBF) are special instances of the decision problem for finite type theory. These instances correspond to formulae in which set membership is not allowed and variables are all taken from the base domain $\mathcal{D}_0$. As we have shown in the previous section, such restricted formulae can be safely encoded in the safe lambda calculus. Therefore since TQBF is PSPACE-complete we have:

Theorem 3.55. *Deciding $\beta\eta$-equality of two safe lambda-terms is PSPACE-hard.* □

Example 3.56. Using the encoding where τ is set to B_0 in the types Δ^n_k for all $k, n \geq 0$, the formula $\forall x \exists y \exists z (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$ is represented by the safe term:

$$\begin{array}{l}
\vdash_{\mathsf{s}} \mathbf{D}^2_0(\lambda x^{\mathsf{B}_2}.AND^0 \\
\quad (\mathbf{D}^1_0(\lambda y^{\mathsf{B}_1}.OR^0 \\
\quad\quad (\mathbf{D}^0_0(\lambda z^{\mathsf{B}_0}.OR^0 \\
\quad\quad\quad (AND^0(OR^0(OR^0\ (\mathbf{C}^{2\mapsto 0}_0\ x)\ (\mathbf{C}^{1\mapsto 0}_0\ y))z) \\
\quad\quad\quad\quad (OR^0(OR^0(NEG^0(\mathbf{C}^{2\mapsto 0}_0\ x))(NEG^0(\mathbf{C}^{1\mapsto 0}_0\ y)))(NEG^0\ z))) \\
\quad\quad)F^0) \\
\quad)F^0) \\
)T^0 \\
: \mathsf{B}_0\ .
\end{array}$$

Remark 3.57 The Boolean satisfaction problem (SAT) is just a particular instance of TQBF where formulae are restricted to use only existential quantifiers, thus the safe lambda calculus is also NP-hard. Asperti gave an interpretation of SAT in the simply-typed lambda calculus but his encoding relies on unsafe terms [Asp].

3.2.4 Other complexity results

3.2.4.1 Better lower bound?

Since the safety condition restricts the expressivity of the lambda calculus in a non-trivial way, one can reasonably expect the beta-eta equality problem (where types are not restricted) to have a lower complexity in the safe case than in the normal case. Our failed attempt to encode type theory in the safe lambda calculus suggests that the non-elementary lower bound that holds in the simply-typed lambda calculus no longer applies in the safe lambda calculus. Nevertheless, one may not rule out the possibility that another non-elementary recursive problem is encodable in the safe lambda calculus.

We have shown that the problem is PSPACE-hard but this is probably a coarse lower bound. It would be interesting to know whether it is also EXPTIME-hard.

3.2.4.2 Upper bound

At present, no upper bound is known for the equivalence problem for safe terms.

3.2.4.3 Beta-eta equivalence for terms limited to a finite set of types

Statman showed [Sta79b] that there exists a finite set of types such that the beta-eta equivalence problem restricted to terms of these types is PSPACE-hard. The picture is different in the safe lambda calculus since our encoding of TQBF requires the full type hierarchy. It was indeed necessary to introduce variables of higher-order in order to eliminate 'unsafety'. Consequently, we had to use simple types of unbounded order. (The order is linear in the size of the QBF formula.) We suspect the decidability problem for safe terms restricted to any finite set of types to have a complexity lower than PSPACE.

3.2.4.4 Normalization

The *normalization problem* is: Given a term M, what is its β-normal form? This problem is non-elementary even when restricted to safe terms as the following example shows. Let $\tau_{-2} \equiv o$ and for $n \geq -1$, $\tau_n \equiv \tau_{n-1} \rightarrow \tau_{n-1}$. For $k, n \in \mathbb{N}$ we write $\overline{k}^n$ to denote the k^{th} Church Numeral parameterized by n as follows:

$$\overline{k}^n \equiv \lambda s^{\tau_{n-1}} z^{\tau_{n-2}}.\overbrace{s(\ldots(s(s}^{k \text{ times}}\, z)\ldots) : \tau_n \ .$$

Then for $n \geq 1$, the safe term $\overline{2}^{n-1}\ \overline{2}^{n-2} \ldots \overline{2}^0$ of type τ_0 has length $\mathcal{O}(n)$ whereas its normal form $\overline{\exp_n(1)}^0$ has length $\mathcal{O}(\exp_n(1))$.

Statman's result shows that in the simply-typed lambda calculus, the beta-eta equality problem is essentially as hard as the normalization problem: they are both non-elementary. It is not known whether this is still the case in the safe lambda calculus. In particular, it may be the case that the beta-eta equivalence problem is elementary although we know that the normalization problem is not.

3.2.4.5 The beta-reduction problem

The *beta-reduction problem* is related to the beta-eta equivalence problem. It can be stated as follows: Given a term M_1 in β-normal form and a term M_2 (possibly containing redexes), does M_2 β-reduce to M_1?

Schubert gave a PSPACE algorithm to decide the β-reduction problem for order-3 lambda-terms [Sch01]. Since order-3 terms are sufficient to encode TQBF in the lambda calculus, this implies that the problem is PSPACE-complete. No complexity result is known for restrictions of this problem to terms of order greater than 3. A natural question is whether complexity characterizations can be obtained when restricting the problem to safe terms.

3.3 Expressivity

3.3.1 Numeric functions representable in the safe lambda calculus

Natural numbers can be encoded in the simply-typed lambda calculus using the Church Numerals: each $n \in \mathbb{N}$ is encoded as the term $\overline{n} = \lambda sz.s^n z$ of type $I = ((o, o), o, o)$ where o is a ground type. We say that a p-ary function $f : \mathbb{N}^p \rightarrow \mathbb{N}$, for $p \geq 0$, is represented by a term $F : (I, \ldots, I, I)$ (with $p + 1$ occurrences of I) if for all $m_i \in \mathbb{N}$, $0 \leq i \leq p$ we have:

$$F\ \overline{m_1} \ldots \overline{m_p} =_\beta \overline{f(m_1, \ldots, m_p)} \ .$$

In 1976 Schwichtenberg [Sch76] showed the following:

Theorem 3.58 (Schwichtenberg 1976). *The numeric functions representable by simply-typed lambda-terms of type $I \rightarrow \ldots \rightarrow I$ using the Church Numeral encoding are exactly the multivariate polynomials* extended with the conditional function.

If we restrict ourselves to safe terms, the representable functions are exactly the multivariate polynomials:

Theorem 3.59. *The functions representable by safe lambda-expressions of type $I \rightarrow \ldots \rightarrow I$ are exactly the multivariate polynomials.*

Proof. Natural numbers are encoded as the Church Numerals: $\overline{n} = \lambda s z. s^n z$ for each $n \in \mathbb{N}$. Addition: For $n, m \in \mathbb{N}$, $\overline{n+m} = \lambda \alpha^{(o,o)} x^o.(\overline{n}\alpha)(\overline{m}\alpha x)$. Multiplication: $\overline{n.m} = \lambda \alpha^{(o,o)}.\overline{n}(\overline{m}\alpha)$. These terms are all safe, furthermore function composition can be safely encoded: suppose that a safe term G of type $I^n \rightarrow I$ represents a function $g : \mathbb{N}^n \rightarrow \mathbb{N}$ and safe terms $F_1, \ldots F_n$ represent functions $f_1, \ldots, f_n : \mathbb{N}^p \rightarrow \mathbb{N}$ then the safe term $\lambda c_1 \ldots c_p.G(F_1 c_1 \ldots c_p) \ldots (F_n c_1 \ldots c_p)$ represents the composed function $(x_1, \cdots, x_p) \mapsto g(f_1(x_1, \ldots, x_p), \ldots, f_n(x_1, \ldots, x_p))$. Hence any multivariate polynomial $P(n_1, \ldots, n_k)$ can be computed by composing the addition and multiplication terms as appropriate.

For the converse, let U be a safe lambda-term of type $I \rightarrow I \rightarrow I$. The generalization to terms of type $I^n \rightarrow I$ for every $n \in \mathbb{N}$ is immediate (They correspond to polynomials with n variables). By Lemma 3.37, safety is preserved by η-long normal expansion therefore we can assume that U is in η-long normal form.

Let $\mathcal{N}^\tau_\Sigma$ denote the set of safe η-long β-normal terms of type τ with free variables in Σ, and $\mathcal{A}^\tau_\Sigma$ for the set of β-normal terms of type τ with free variables in Σ and of the form $\varphi s_1 \ldots s_m$ for some variable $\varphi : (A_1, \ldots, A_m, o)$ where $m \geq 0$ and for all $1 \leq i \leq m$, $s_i \in \mathcal{N}^{A_i}_\Sigma$. Observe that the set $\mathcal{A}^o_\Sigma$ contains only safe terms but the sets $\mathcal{A}^\tau_\Sigma$ in general may contain unsafe terms. Let Σ denote the alphabet $\{x, y : I, z : o, \alpha : o \rightarrow o\}$. By an easy reasoning (See the term grammar construction of Zaionc [Zai87]), we can derive the following equations inducing a grammar over the set of terminals $\Sigma \cup \{\lambda x y \alpha z., \lambda z.\}$ that generates precisely the terms of $\mathcal{N}^{(I,I,I)}_\emptyset$:

$$
\begin{array}{rcl}
\mathcal{N}^{(I,I,I)}_\emptyset & \rightarrow & \lambda x y \alpha z.\mathcal{A}^o_\Sigma \\
\mathcal{A}^o_\Sigma & \rightarrow & z \mid \mathcal{A}^{(o,o)}_\Sigma \mathcal{A}^o_\Sigma \\
\mathcal{A}^{(o,o)}_\Sigma & \rightarrow & \alpha \mid \mathcal{A}^I_\Sigma\, \mathcal{N}^{(o,o)}_\Sigma \\
\mathcal{N}^{(o,o)}_\Sigma & \rightarrow & \lambda z.\mathcal{A}^o_\Sigma \\
\mathcal{A}^I_\Sigma & \rightarrow & x \mid y \ .
\end{array}
$$

The key rule is the fourth one: had we not imposed the safety constraint the right-hand side would instead be of the form $\lambda w^o.\mathcal{A}^{(o,o)}_{\Sigma \cup \{w:o\}}$. Here the safety constraint imposes to abstract all the ground type variables occurring freely, thus only one free variable of ground type can appear in the term and we can choose it to be named z up to α-conversion.

We extend the notion of representability to terms of type o, (o,o) and I with free variables in Σ as follows: A function $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ is represented by a term $\Sigma \vdash_{\mathsf{st}} F : o$ if and only if for all $m, n \in \mathbb{N}$, $F[\overline{m}, \overline{n}/x, y] =_\beta \alpha^{\overline{f(m,n)}} z$; by a term $\Sigma \vdash_{\mathsf{st}} G : (o,o)$ iff $G[\overline{m}, \overline{n}/x, y] =_\beta \lambda z.\alpha^{\overline{f(m,n)}} z$; and by $\Sigma \vdash_{\mathsf{st}} H : I$ iff $H[\overline{m}, \overline{n}/x, y] =_\beta \lambda \alpha z.\alpha^{\overline{f(m,n)}} z$.

We now show by induction on the grammar rules that any term generated by the grammar represents some polynomial: *Base case:* The term x and y represent the projection functions $(m, n) \mapsto m$ and $(m, n) \mapsto n$ respectively. The term α and z represent the constant functions $(m, n) \mapsto 1$ and $(m, n) \mapsto 0$ respectively. *Step case:* The first and fourth rule are trivial: for $F \in \mathcal{A}^o_\Sigma$, $\lambda z.F$ and $\lambda x y \alpha z.F$ represent the same function as F. We now consider the second and third rule. We observe that for $m, p, p' \geq 0$ we have

$$(i)\ \ \overline{m}\,(\lambda z.\alpha^p z) =_\beta \lambda z.\alpha^{m \cdot p} z; \qquad\qquad (ii)\ \ (\lambda z.\alpha^p z)(\alpha^{p'} z) =_\beta \alpha^{p+p'} z\ .$$

Suppose that $F \in \mathcal{A}^I_\Sigma$ and $G \in \mathcal{N}^{(o,o)}_\Sigma$ represent the functions f and g respectively then by (i), FG represents the function $f \times g$. And if $F \in \mathcal{A}^{(o,o)}_\Sigma$ and $G \in \mathcal{N}^o_\Sigma$ represent the functions f and g then by (ii), FG represents the function $f + g$.

Hence U represents some polynomial: for all $m, n \in \mathbb{N}$ we have $U\ \overline{m}\ \overline{n} =_\beta \lambda\alpha z.\alpha^{p(m,n)}z$ where $p(m,n) = \sum_{0 \leq k \leq d} m^{i_k} n^{j_k}$ for some $i_k, j_k \geq 0$, $d \geq 0$. ☐

Corollary 3.60. *The conditional operator $C : I \rightarrow I \rightarrow I \rightarrow I$ satisfying:*

$$C\ t\ y\ z \rightarrow_\beta \begin{cases} y, & \text{if } t \rightarrow_\beta \overline{0}\ ; \\ z, & \text{if } t \rightarrow_\beta \overline{n+1}\ . \end{cases}$$

is not definable in the simply-typed safe lambda calculus.

Example 3.61. The term $\lambda FGH\alpha x.F(\underline{\lambda y.G\alpha x})(H\alpha x)$ used by Schwichtenberg [Sch76] to define the conditional operator is unsafe since the underlined subterm, which is of order 1, occurs at an operand position and contains an occurrence of x of order 0.

Remark 3.62

1. This corollary tells us that the conditional function is not definable when numbers are represented by the Church Numerals. It may still be possible, however, to represent the conditional function using a different encoding for natural numbers. One way to compensate for the loss of expressivity caused by the safety constraint is to introduce countably many domains of representation for natural numbers. Such a technique is used to represent the predecessor function in the simply-typed lambda calculus [FLO83].

2. There are other ways to interpret conditional in the lambda calculus. For instance the (unsafe) lambda-term $\lambda txy.(C\,t\,\overline{0}\,\overline{1})(\lambda u.\underline{y})x$ of type $I \rightarrow o \rightarrow o \rightarrow o$ behaves like the conditional operator C. It can be shown that there is no such term in the safe lambda calculus simply because the only safe terms of type $I \rightarrow o \rightarrow o \rightarrow o$ up to $\alpha\beta\eta$-equivalence are $\lambda txy.x$ and $\lambda txy.y$.

3. The boolean conditional can be represented in the safe lambda calculus as follows: We encode booleans by terms of type $B = (o,o,o)$. The two truth values are then represented by $\lambda x^o y^o.x$ and $\lambda x^o y^o.y$; the conditional is then given by $\lambda F^B G^B H^B x^o y^o.F\,(G\,x\,y)(H\,x\,y)$.

4. It is also possible to define a conditional operator behaving like the conditional operator C in the second-order lambda calculus [FLO83]: natural numbers are represented by terms $\overline{n} \equiv \Lambda t.\lambda s^{t\rightarrow t} z^t.s^n(z)$ of type $J \equiv \Delta t.(t \rightarrow t) \rightarrow (t \rightarrow t)$ and the conditional is encoded by the term $\lambda F^J G^J H^J.F\ J\ (\lambda u^J.G)\ H$. Whether this term is safe or not cannot be answered just yet as we do not have a notion of safety for second-order typed terms.

3.3.2 Word functions definable in the safe lambda calculus.

Schwichtenberg's result on numeric functions definable in the lambda calculus was extended to richer structures: Zaionc studied the problem for word functions, then functions over trees and eventually the general case of functions over free algebras [Lei93, Zai91, Zai88, Zai87, Zai95]. In this section we consider the case of word functions expressible in the safe lambda calculus.

Word functions.

We consider a binary alphabet $\Sigma = \{a, b\}$. The result of this section naturally extends to all finite alphabets. We consider the set Σ^* of all words over Σ. The empty words is denoted ϵ. We write $|w|$ to denote the length of the word $w \in \Sigma^*$. For any $k \in \mathbb{N}$ we write $\mathbf{k}$ to denote the word $a \ldots a$ with k occurrences of a, so that $|\mathbf{k}| = k$. For any $n \geq 1$ and $k \geq 0$, we write $c(n,k)$ for the n-ary function $(\Sigma^*)^n \rightarrow \Sigma^*$ that maps all inputs to the word $\mathbf{k}$. We consider various word functions. Let x, y, z be words over Σ:

- Concatenation $app : (\Sigma^*)^2 \to \Sigma^*$. The word $app(x, y)$ is the concatenation of x and y.
- Substitution $sub : (\Sigma^*)^3 \to \Sigma^*$. The word $sub(x, y, z)$ is obtained from x by substituting the word y for all occurrences of a and z for all occurrences of b. Formally:

$$\begin{aligned} sub(\epsilon, y, z) &= \epsilon \ , \\ sub(ax, y, z) &= app(y, sub(x, y, z)) \ , \\ sub(bx, y, z) &= app(z, sub(x, y, z)) \ . \end{aligned}$$

- Prefix-cut $cut_a : \Sigma^* \to \Sigma^*$. The word $cut_a\, x$ is the maximal prefix of x containing only the letter 'a'. Formally:

$$\begin{aligned} cut_a(\epsilon) &= \epsilon \ , \\ cut_a(ax) &= app(a, cut_a(x)) \ , \\ cut_a(bx) &= \epsilon \ . \end{aligned}$$

- Projections $\pi_k : (\Sigma^*)^n \to \Sigma^*$ for $n \geq 1$, $1 \leq k \leq n$ defined as $\pi_k(x_1, \ldots, x_k, \ldots, x_n) = x_k$.
- Constant functions $cst_w : \Sigma^* \to \Sigma^*$ for $w \in \Sigma^*$, mapping constantly onto the word w.

Additional operations can be obtained by combining the above functions [Zai91]:

- Prefix-cut $cut_b : \Sigma^* \to \Sigma^*$ is defined by $cut_b(x) = sub(cut_a(sub(x, b, a)), b, a)$.
- Non-emptiness check $\overline{sq} : \Sigma^* \to \Sigma^*$ (returns **0** if the word is ϵ and **1** otherwise) is defined by $\overline{sq}(x) = cut_a(app(sub(x, b, b), a)$.
- Emptiness check $sq : \Sigma^* \to \Sigma^*$ is defined by $sq(x) = \overline{sq}(\overline{sq}(x))$.
- Occurrence check $occ_l : \Sigma^* \to \Sigma^*$ of the letter $l \in \Sigma$ (returns **1** if the word contains an occurrence of l and **0** otherwise) is defined by $occ_l(x) = sq(sub(x, l, \epsilon))$.

Representability

We consider equality of terms modulo α, β and η conversion, and we write $M =_{\beta\eta} N$ to denote this equality. For every simple type τ, we write $\mathrm{Cl}(\tau)$ for the set of closed terms of type τ (modulo α, β and η conversion).

Take the type $\mathbf{B} = (o \to o) \to (o \to o) \to o \to o$, called *the binary word type* [Zai87]. There is a 1-1 correspondence between words over Σ and closed terms of type $\mathbf{B}$. Think of the first two parameters as concatenators for 'a' and 'b' respectively, and the third parameter as the constructor for the empty word. Thus the empty word ϵ is represented by $\lambda u^{o\to o} v^{o \to o} x^o.x$; if $w \in \Sigma^*$ is represented by a term $W \in \mathrm{Cl}(\mathbf{B})$ then $a \cdot w$ is represented by $\lambda u^{o\to o} v^{o\to o} x^o.u(Wuvx)$ and $b \cdot w$ is represented by $\lambda u^{o\to o} v^{o\to o} x^o.v(Wuvx)$. For any word $w \in \Sigma^*$ we write $\underline{w}$ to denote the term representation obtained that way. We say that the word function $h : (\Sigma^*)^n \to \Sigma^*$ is ***represented*** by a closed term $H \in \mathrm{Cl}(\mathbf{B}^n \to \mathbf{B})$ just if for all $x_1, \ldots, x_n \in \mathbf{B}^*$, $H\underline{x_1} \ldots \underline{x_n} =_{\beta\eta} \underline{h x_1 \ldots x_n}$.

Example 3.63. The word functions $app, sub, cut_a, cut_b, sq, \overline{sq}, occ_a, occ_b$ defined above are respectively represented by the following lambda-terms:

$$\begin{aligned} \mathrm{APP} &\equiv \lambda cduvx.cuv(duvx), & \mathrm{SUB} &\equiv \lambda xdeuvx.c(\lambda y.duvy)(\lambda y.euvy)x, \\ \mathrm{CUT}_a &\equiv \lambda cuvx.cu(\lambda y.x)x, & \mathrm{CUT}_b &\equiv \lambda cuvx.c(\lambda y.x)vx, \\ \mathrm{SQ} &\equiv \lambda cuvx.c(\lambda y.ux)(\lambda y.ux)x, & \overline{\mathrm{SQ}} &\equiv \lambda cuvx.c(\lambda y.x)(\lambda y.x)(ux), \\ \mathrm{OCC}_a &\equiv \lambda cuvx.c(\lambda y.ux)(\lambda y.y)x, & \mathrm{OCC}_b &\equiv \lambda cuvx.c(\lambda y.y)(\lambda y.ux)x. \end{aligned}$$

Zaionc [Zai87] showed that the λ-definable word functions are generated by a finite base in the following sense:

Theorem 3.64 (Zaionc [Zai87])**.** *The set of λ-definable word functions is the minimal set containing: (i) the constant functions; (ii) the projections; (iii) concatenation app; (iv) substitution sub; (v) prefix-cut cut_a; and closed by composition.*

The terms representing these basic operations are given in Example 3.63. We observe that among them, only APP and SUB are safe; the other terms are all unsafe because they contain terms of the form $N(\lambda y.x)$ where x and y are of the same order. It turns out that APP and SUB constitute a base of terms generating all the functions definable in the safe lambda calculus as the following theorem states:

Theorem 3.65. *Let* $\lambda^{\text{safe}}\text{def}$ *denote the minimal set containing the following word functions and closed by composition:*
(i) the projections;
(ii) the constant functions;
(iii) concatenation app;
(iv) substitution sub.
The set of word functions definable in the safe lambda calculus is precisely $\lambda^{\text{safe}}\text{def}$.

The proof follows the same steps as Zaionc's proof. The first direction is immediate: Projections are represented by safe terms of the form $\lambda x_1 \ldots x_n.x_i$ for some $i \in \{1..n\}$, and constant functions by $\lambda x_1 \ldots x_n.\underline{w}$ for some $w \in \Sigma^*$. The terms APP and SUB are safe and represent concatenation and substitution. For closure by composition: take a function $g : (\Sigma^*)^n \rightarrow \Sigma^*$ represented by safe term $G \in \mathrm{Cl}(\mathbf{B}^n \rightarrow \mathbf{B})$ and functions $f_1, \ldots, f_n : (\Sigma^*)^p \rightarrow \Sigma^*$ represented by safe terms $F_1, \ldots F_n$ respectively then the function

$$(x_1, \cdots, x_p) \mapsto g(f_1(x_1, \ldots, x_p), \ldots, f_n(x_1, \ldots, x_p))$$

is represented by the term $\lambda c_1 \ldots c_p.G(F_1 c_1 \ldots c_p) \ldots (F_n c_1 \ldots c_p)$ which is also safe.

To show the other direction we need to introduce some more definitions. We will write $\mathrm{Op}(n, k)$ to denote the set of open terms M typable as follows:

$$c_1 : \mathbf{B}, \ldots c_n : \mathbf{B}, u : (o, o), v : (o, o), x_{k-1} : o, \ldots, x_0 : o \vdash_{\mathsf{st}} M : o \ .$$

Thus we have the following equality (modulo α, β and η conversions) for $n, k \geq 1$:

$$\mathrm{Cl}(\tau(n, k)) = \{\lambda c_1^{\mathbf{B}} \ldots c_n^{\mathbf{B}} u^{(o,o)} v^{(o,o)} x_{k-1}^o \ldots x_0^o.M \mid M \in \mathrm{Op}(n, k)\}$$

writing $\tau(n, k)$ as a shorthand for the type $\mathbf{B}^n \rightarrow (o, o)^2 \rightarrow o^k \rightarrow o$. We generalize the notion of representability to terms of type $\tau(n, k)$ as follows:

Definition 3.66 (Function pair representation)**.** A closed term $T \in \mathrm{Cl}(\tau(n, k))$ ***represents the pair of functions*** (f, p) where $f : (\Sigma^*)^n \rightarrow \Sigma^*$ and $p : (\Sigma^*)^n \rightarrow \{\mathbf{0}, \ldots, \mathbf{k-1}\}$ if for all $w_1, \ldots, w_n \in \Sigma^*$ and for every $i \in \{0 \ldots, k-1\}$ we have:

$$T\underline{w_1} \ldots \underline{w_n} =_{\beta\eta} \lambda uvx_{k-1} \ldots x_0.\underline{f(w_1, \ldots, w_n)}uvx_{|p(w_1, \ldots, w_n)|} \ .$$

By extension we will say that an *open* term M from $\mathrm{Op}(n, k)$ represents the pair (f, p) just if $M[\underline{w_1} \ldots \underline{w_n}/c_1 \ldots c_n] =_{\beta\eta} \underline{f(w_1, \ldots, w_n)}uvx_{|p(w_1, \ldots, w_n)|}$.

We will call ***safe pair*** any pair of functions of the form $(w, c(n, i))$ where $0 \leq i \leq k-1$ and w is an n-ary function from $\lambda^{\text{safe}}\text{def}$.

Theorem 3.67 (Characterization of the representable pairs)**.** *The function pairs representable in the safe lambda calculus are precisely the safe pairs.*

Proof. (Soundness). Take a pair $(w, c(n,i))$ where $0 \leq i \leq k-1$ and w is an n-ary function from λ^{safe}def. As observed earlier, all the functions from λ^{safe}def are representable in the safe lambda calculus: Let $\underline{w}$ be the representative of w. The pair $(w, c(n,i))$ is then represented by the term $\lambda c_1 \ldots c_n u v x_{k-1} \ldots x_0 . \underline{w} c_1 \ldots c_n u v x_i$.

(Completeness) It suffices to consider safe β-η-long normal terms from $\mathrm{Op}(n,k)$ only. The result then follows immediately for every safe term in $\mathrm{Cl}(\tau(n,k))$. The subset of $\mathrm{Op}(n,k)$ consisting of β-η-long normal terms is generated by the following grammar [Zai87]:

$$
\begin{array}{lll}
(\alpha_i^k) & R^k & \rightarrow x_i \\
(\beta^k) & & \mid u R^k \\
(\gamma^k) & & \mid v R^k \\
(\delta_j^k) & & \mid c_j \ \overbrace{(\lambda z^k . R^{k+1}[z^k, x_0, \ldots, x_{k-1}/x_0, x_1, \ldots, x_k])}^{Q^k(R^{k+1})} \\
& & \quad (\lambda z^k . R^{k+1}[z^k, x_0, \ldots, x_{k-1}/x_0, x_1, \ldots, x_k]) \\
& & \quad R^k
\end{array}
$$

for $k \geq 1$, $0 \leq i < k$, $0 \leq j \leq n$. The notation $M[\ldots/\ldots]$ denotes the usual simultaneous substitution. The non-terminals are R^k for $k \geq 1$ and the set of terminals is $\{z^k, \lambda z^k \mid k \geq 1\} \cup \{x_i \mid i \geq 0\} \cup \{c_1, \ldots, c_n, u, v\}$.

The name of each rule is indicated in parenthesis. We identify a rule name with the right-hand side of the rule, thus α_i^k belongs to $\mathrm{Op}(n,k)$, β^k and γ^k are functions from $\mathrm{Op}(n,k)$ to $\mathrm{Op}(n,k)$, and δ_j^k is a function from $\mathrm{Op}(n,k+1) \times \mathrm{Op}(n,k+1) \times \mathrm{Op}(n,k)$ to $\mathrm{Op}(n,k)$.

We now want to characterize the subset consisting of all *safe* terms generated by this grammar. The term α_i^k is always safe; $\beta^k(M)$ and $\gamma^k(M)$ are safe if and only if M is; and $\delta_j^k(F,G,H)$ is safe if and only if $Q^k(F)$, $Q^k(G)$ and H are safe. The free variables of $Q^k(F)$ belong to $\{c_1, \ldots c_n, u, v, x_0, \ldots x_k\}$ thus they have order greater than $\operatorname{ord} z$ except the x_is which have the same order as z. Hence since the x_is are not abstracted together with z we have that $Q^k(F)$ is safe if and only if F is safe and the variables $x_0 \ldots x_k$ do not appear free in $F[z^k, x_0, \ldots, x_{k-1}/x_0, x_1, \ldots, x_k]$, or equivalently if the variables $x_1 \ldots x_k$ do not appear free in F. Similarly, $Q^k(G)$ is safe if and only if G is safe and the variables $x_1 \ldots x_k$ do not appear free in G.

We therefore need to identify the subclass of terms generated by the non-terminal R^k which are safe and which do not have any free occurrence of variables in $\{x_1 \ldots x_{k-1}\}$. By imposing this requirement to the rules of the previous grammar we obtain the following specialized grammar characterizing the desired subclass:

$$
\begin{array}{lll}
(\overline{\alpha}_0^k) & \overline{R}^k & \rightarrow x_0 \\
(\overline{\beta}^k) & & \mid u \overline{R}^k \\
(\overline{\gamma}^k) & & \mid v \overline{R}^k \\
(\overline{\delta}_j^k) & & \mid c_j \ (\lambda z^k . \overline{R}^{k+1}[z^k/x_0]) \ (\lambda z^k . \overline{R}^{k+1}[z^k/x_0]) \ \overline{R}^k \ .
\end{array}
$$

For every term M, $Q^k(M)$ is safe if and only if M can be generated from the non-terminal $\overline{R}^k$. Thus the subset of $\mathrm{Cl}(\tau(n,k))$ consisting of safe beta-normal terms is given by the grammar:

$$
\begin{array}{lll}
(\widetilde{\pi}^k) & \widetilde{S} & \rightarrow \lambda c_1 \ldots c_n u v x_{k-1} \ldots x_0 . \widetilde{R}^k \\
(\widetilde{\alpha}_i^k) & \widetilde{R}^k & \rightarrow x_i \\
(\widetilde{\beta}^k) & & \mid u \widetilde{R}^k \\
(\widetilde{\gamma}^k) & & \mid v \widetilde{R}^k \\
(\widetilde{\delta}_j^k) & & \mid c_j \ (\lambda z^k . \overline{R^{k+1}}[z^k/x_0]) \ (\lambda z^k . \overline{R^{k+1}}[z^k/x_0]) \ \widetilde{R}^k \ .
\end{array}
$$

To conclude the proof it thus suffices to show that every term generated by this grammar (starting with the non-terminal $\widetilde{S}$) represents a safe pair. We proceed by induction and show that the non-terminal $\overline{R}^k$ generates terms representing pairs of the form $(w, c(n,0))$ while non-terminals $\widetilde{S}$ and $\widetilde{R}^k$ generate terms representing pairs of the form $(w, c(n,i))$ for $0 \leq i < k$ and $w \in \lambda^{\text{safe}}\text{def}$.

Base case: The term $\overline{\alpha}^k_0$ represents the safe pair $(c(n,0), c(n,0))$ while $\widetilde{\alpha}^k_i$ represents the safe pair $(c(n,0), c(n,i))$. *Step case:* Suppose $T \in \mathrm{Op}(n,k)$ represents a pair (w,p). Then $\overline{\beta}^k(T)$ and $\widetilde{\beta}^k(T)$ represent the pair $(app(a,w),p)$; $\overline{\gamma}^k(T)$ and $\widetilde{\gamma}^k(T)$ represent the pair $(app(b,w),p)$; and $\overline{\pi}^k(T) \in \mathrm{Cl}(\tau(n,k))$ represents the pair (w,p). Now suppose that E, F and G represent the pairs $(w_e, c(n,0))$, $(w_f, c(n,0))$ and $(w_g, c(n,i))$ respectively. Then we have:

$$
\begin{aligned}
&\widetilde{\delta}^k_j(E,F,G)[\underline{w_1} \ldots \underline{w_n}/c_1 \ldots c_n] \\
&\qquad = \underline{w_j}\ (\lambda z^k.E[z^k/x_0])[\underline{w_1} \ldots \underline{w_n}/c_1 \ldots c_n] \\
&\qquad\qquad (\lambda z^k.F[z^k/x_0])[\underline{w_1} \ldots \underline{w_n}/c_1 \ldots c_n] \\
&\qquad\qquad G[\underline{w_1} \ldots \underline{w_n}/c_1 \ldots c_n] \\
&\qquad =_{\beta\eta} \underline{w_j}\ (\lambda z^k.E[\underline{w_1} \ldots \underline{w_n}/c_1 \ldots c_n][z^k/x_0]) \\
&\qquad\qquad (\lambda z^k.F[\underline{w_1} \ldots \underline{w_n}/c_1 \ldots c_n][z^k/x_0]) \\
&\qquad\qquad (\underline{w_g(w_1 \ldots w_n)}\ u\ v\ x_i) && G \text{ represents } (h, c(n,i)) \\
&\qquad =_{\beta\eta} \underline{w_j}\ (\lambda z^k.(\underline{w_e(w_1 \ldots w_n)}\ u\ v\ x_0)[z^k/x_0]) && E \text{ represents } (f, c(n,0)) \\
&\qquad\qquad (\lambda z^k.(\underline{w_f(w_1 \ldots w_n)}\ u\ v\ x_0)[z^k/x_0]) && F \text{ represents } (g, c(n,0)) \\
&\qquad\qquad (\underline{w_g(w_1 \ldots w_n)}\ u\ v\ x_i) \\
&\qquad =_{\beta\eta} \underline{w_j}\ (\lambda z^k.\underline{w_e(w_1 \ldots w_n)}\ u\ v\ z^k) \\
&\qquad\qquad (\lambda z^k.\underline{w_f(w_1 \ldots w_n)}\ u\ v\ z^k) \\
&\qquad\qquad (\underline{w_g(w_1 \ldots w_n)}\ u\ v\ x_i) \\
&\qquad =_{\eta} \underline{w_j}\ (\underline{w_e(w_1 \ldots w_n)}\ u\ v)\ (\underline{w_f(w_1 \ldots w_n)}\ u\ v)\ (\underline{w_g(w_1 \ldots w_n)}\ u\ v\ x_i) \\
&\qquad =_{\beta\eta} \underline{w}\ u\ v\ x_i
\end{aligned}
$$

where the word function w is defined as

$$w : w_1, \ldots, w_n \mapsto app(sub(w_j, w_e(w_1, \ldots, w_n), w_f(w_1, \ldots, w_n)), w_g(x_1, \ldots, w_n)) \ .$$

Hence $\widetilde{\delta}^k_j(E,F,G)$ represents the pair $(w, c(n,i))$.

The same argument shows that if E, F and G all represent safe pairs then so does $\overline{\delta}^k_j(E,F,G)$. $\square$

Theorem 3.65 is obtained by instantiating Theorem 3.67 with terms of types $\tau(n,1) = I^n \rightarrow I$: every closed safe term of this type represents some n-ary function from $\lambda^{\text{safe}}\text{def}$.

3.4 Typing problems

In this section we consider the problems of type checking, typability and type inhabitation as defined in Sec. 2.1 but recast in the safe lambda calculus:

- Type checking: Given a term M, context Γ and type A, do we have $\Gamma \vdash_{\mathsf{s}} M : A$?
- Typability: Given a term M and context Γ, is there a type A such that $\Gamma \vdash_{\mathsf{s}} M : A$?
- Inhabitation: Given a type A, is there a term M such that $\vdash_{\mathsf{s}} M : A$?

We will restrict our attention to the Church-like safe lambda calculus. The results presented here straightforwardly extend to the Curry version.

3.4.1 Relating derivations from $\Lambda_{\rightarrow}^{\mathrm{Cu}}$ and safe $\Lambda_{\rightarrow}^{\mathrm{Cu}}$

In this section we compare derivations obtained in the simply-typed lambda calculus with those obtained in the safe lambda calculus. In order to ease the comparison, we introduce an alternative presentation of the simply-typed lambda calculus. The rules of this typing system are given in Table 3.4. There are two main differences with the rules of Def. 2.14: (i) There is now a weakening rule; (ii) Simultaneous consecutive applications and abstractions can be performed at once.

$$\frac{}{x : A \vdash_{\mathrm{Cu}} x : A} \qquad \frac{\Gamma \vdash_{\mathrm{Cu}} M : A}{\Delta \vdash_{\mathrm{Cu}} M : A}\ \Gamma \subset \Delta$$

$$\frac{\Gamma \vdash_{\mathrm{Cu}} M : (A_1, \ldots, A_n, B) \quad \Gamma \vdash_{\mathrm{Cu}} N_1 : A_1 \quad \ldots \quad \Gamma \vdash_{\mathrm{Cu}} N_n : A_n}{\Gamma \vdash_{\mathrm{Cu}} M N_1 \ldots N_n : B}$$

$$\frac{\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash_{\mathrm{Cu}} M : B}{\Gamma \vdash_{\mathrm{Cu}} \lambda x_1 \ldots x_n . M : (A_1, \ldots, A_n, B)}$$

Table 3.4: Alternative definition of the lambda calculus *à la* Curry.

The two presentations are clearly equivalent in the sense that $\Gamma \vdash_{\mathrm{Cu}} M : T$ is derivable in this system iff it is derivable with the rules of Def. 2.14.

Convention 3.68 In order to make our derivations canonical, we adopt the following convention:

- A derivation cannot contain two consecutive applications of the weakening rule;
- when using the weakening rule, the context Δ is chosen as small as possible so that for every judgement $\Gamma \vdash_{\mathrm{Cu}} M : A$ appearing in the derivation that is not deduced from the weakening rule we have $FV(M) = \mathrm{dom}(\Gamma)$.

We are interested in derivations satisfying the following property: A deduction Δ of $\Gamma \vdash_{\mathrm{Cu}} \underset{\sim}{M} : T$ is ***compact*** if the set of terms appearing in the nodes of the deduction tree Δ is precisely $\mathrm{sub}(M)$. In other words in a compact deduction, each use of the application and abstraction rule in the deduction is as "large" as possible so that each path in the deduction tree consists of an axiom followed by an alternation of application/abstraction rules. Compact derivations are sufficient: if there is derivation in $\Lambda_{\rightarrow}^{\mathrm{Cu}}$ then there is a compact derivation with the same conclusion. We will write $\mathsf{Der}_{cu}(\Gamma, M, T)$ for the set of compact derivations of $\Gamma \vdash_{\mathrm{Cu}} M : T$.

Similarly, we define the notion of compact derivation in the safe lambda calculus. It is easy to check that, despite the side-conditions imposed by the abstraction rule, the compact deductions are sufficient. We write $\mathsf{Der}_{s}(\Gamma, M, T)$ for the set of compact deductions of $\Gamma \vdash_{\mathsf{s}} M : T$ in safe $\Lambda_{\rightarrow}^{\mathrm{Cu}}$.

We say that a deduction $\Delta \in \mathsf{Der}_{cu}(\Gamma, M, T)$ is ***safe*** if $\mathrm{ord}\,\Gamma \geq \mathrm{ord}\,T$ and for every term-in-context $\Gamma' \vdash_{\mathsf{st}} M : T'$ from Δ that is deduced using the abstraction rule we have $\mathrm{ord}\,\Gamma' \geq \mathrm{ord}\,T'$.

For every deduction tree Δ in $\mathsf{Der}_{s}(\Gamma, M, T)$ we write $\epsilon(\Delta)$ to denote the deduction tree obtained by replacing judgements $\Gamma \vdash_{\mathsf{s}} M : T$ by $\Gamma \vdash_{\mathrm{Cu}} M : T$ and rules of the safe lambda calculus by their counterpart in the simply-typed lambda calculus (identifying (app) and $(\mathsf{app_{as}})$).

Lemma 3.69 (Relating derivations from $\Lambda_{\rightarrow}^{\mathrm{Cu}}$ and safe $\Lambda_{\rightarrow}^{\mathrm{Cu}}$).

(i) $\Delta \in \mathsf{Der}_{s}(\Gamma, M, T) \implies \epsilon(\Delta) \in \mathsf{Der}_{cu}(\Gamma, M, T) \wedge \epsilon(\Delta)$ *is safe,*

(ii) $\Delta' \in \mathsf{Der}_{cu}(\Gamma, M, T) \wedge \Delta'$ *is safe.* $\implies \exists \Delta \in \mathsf{Der}_{s}(\Gamma, M, T) : \Delta' = \epsilon(\Delta)$.

Proof. This follows immediately from the definition of safe $\Lambda_{\rightarrow}^{\mathrm{Cu}}$. ∎

3.4.2 Type checking and typability

By the Principal Type (PT) Theorem 2.20, if a term is typable then it has a computable principal derivation: every other derivation is an instance of that derivation. The same result holds for compact derivations:

Lemma 3.70 (Principal compact derivation)**.** *If M is typable in $\Lambda^{\mathrm{Cu}}_{\rightarrow}$ then it has a* compact *principal derivation Δ (i.e., any derivation $\Delta' \in \mathsf{Der}_{cu}(\Gamma, M, T)$ is an instance of Δ) that is computable from M.*

Proof. This follows immediately from Theorem 2.20. Compact derivations are just "reorganized" derivations: for every standard derivation there exists a corresponding compact derivation containing the same typing assumptions. The *compact* principal derivations can be obtained from the principal derivation by performing the very same "reorganization". ☐

Proposition 3.71. TYPE CHECKING *in safe $\Lambda^{\mathrm{Cu}}_{\rightarrow}$ is decidable.*

Proof. Let $M \in \Lambda$, $T \in \mathbb{T}$ and Γ be a typing-context. We have $\Gamma \vdash_{\mathsf{s}} M : T$ iff $\mathsf{Der}_s(\Gamma, M, T) \neq \emptyset$. By Lemma 3.69, there is a derivation in $\mathsf{Der}_s(\Gamma, M, T)$ if and only if there is a safe derivation in $\mathsf{Der}_{cu}(\Gamma, M, T)$. We already know that the TYPE CHECKING problem in $\Lambda^{\mathrm{Cu}}_{\rightarrow}$ ("Is $\mathsf{Der}_{cu}(\Gamma, M, T)$ empty?") is decidable. If $\mathsf{Der}_{cu}(\Gamma, M, T)$ is empty then we can answer 'No' to the type-checking problem. Otherwise by the previous Lemma, we can compute a compact principal derivation Δ_p of $\Gamma \vdash_{\mathsf{s}} M : T$ and we know that there exists a safe derivation iff there exists a type-substitution s for Δ_p such that (i) $s(\Delta_p)$ is safe; (ii) the conclusion of $s(\Delta_p)$ is $\Gamma \vdash_{\mathsf{s}} M : T$.

The latter property can be decided by unifying the types appearing in the conclusion of Δ_p with Γ and T. The former property turns out to be also decidable. Indeed, the deduction Δ_p contains finitely many atoms $a_1 \ldots a_n \in \mathbb{A}$, $n \geq 1$. Therefore the safety of $s(\Delta_p)$ can be expressed in terms of a system of inequations over the order of the atoms occurring in Δ_p. This system can be reexpressed into a system of inequations $\mathcal{S}$ of the form $x_i > x_j$ for $i, j \in \{1, .., q\}$ and variables $x_1, \ldots, x_q \in \mathbb{Z}$ and such that for every atom a_k, $\operatorname{ord} a_k = x_{i_k}$ for some $i_k \in \{1, .., q\}$.

A substitution s satisfying the required property exists if and only if $\mathcal{S}$ has a solution. If the solution to $\mathcal{S}$ is $(x_1, \ldots, x_q)$ then we take the substitution $s = [(x_{k_1})_o/a_1, \ldots (x_{k_n})_o/a_n]$ for some fresh atom $o \in \mathbb{A}$. (Observe that if $(x_1, \ldots, x_q)$ is a solution then so is $(x_1 + k, \ldots, x_q + k)$ for $k \geq 0$, therefore the x_is can all be assumed to be positive.) The system $\mathcal{S}$ can then be solved using a topological sorting algorithm [Knu00]. ☐

Proposition 3.72. TYPABILITY *in safe $\Lambda^{\mathrm{Cu}}_{\rightarrow}$ is decidable.*

Proof. The proof is the same as for TYPE CHECKING except that only condition (i) needs to be decided. ☐

3.4.3 The type inhabitation problem

Statman showed that the problem of deciding whether a type *defined over an infinite number of ground atoms* is inhabited (or equivalently of deciding validity of an intuitionistic implicative formula) is PSPACE-complete [Sta79a]. In the safe lambda calculus, no complexity is known. In fact it is not even clear whether the problem is decidable:

Proposition 3.73. INHABITATION *in safe $\Lambda_{\rightarrow}$ is (at least) semi-decidable: Given a simple type, there is an algorithm that prints out a safe inhabitant if there is one but may not terminate if there is not.*

Proof. Inhabitants are enumerated using Ben-Yelles's counting algorithm [Hin97] and each inhabitant can be tested for typability in safe $\Lambda_{\rightarrow}$ by Proposition 3.72. ☐

It is well known that the simply-typed lambda calculus corresponds to intuitionistic implicative logic via the Curry-Howard isomorphism. The theorems of the logic correspond to inhabited types; further every inhabitant of a type represents a proof of the corresponding formula. Similarly, we can consider the fragment of intuitionistic implicative logic that corresponds to the safe lambda calculus under the Curry-Howard isomorphism; we call it the *safe fragment of intuitionistic implicative logic*.

We would like to compare the reasoning power of these two logics, in other words, to determine which types are inhabited in the lambda calculus but not in the safe lambda calculus.[1] Since safety is preserved by β-reduction, it is enough to look at *normal inhabitants*—those inhabitants that are in β-normal form. We say that a type is ***unsafe*** if it is inhabited and every inhabitant is unsafe. At order 2, all closed normal terms are safe therefore there is no unsafe type at this order. The following proposition further shows that every type generated from a single atom o is not unsafe:

Proposition 3.74. *Every type generated from one atom o that is inhabited in the lambda calculus is also inhabited by a safe lambda-term.*

Proof. One can transform any unsafe normal inhabitant M into a safe one of the same type as follows: Compute the eta-long beta-normal form of M. Let x be an occurrence of a ground-type variable in a subterm of the form $\lambda\overline{x}.C[x]$ where $\lambda\overline{x}$ is the binder of x and for some context $C[-]$ different from the identity ($C[-] \equiv -$). Since the term is beta-normal and because its type is built out of a unique atom o, x is necessarily of type o. We then replace the subterm $C[x]$ by x in M. This transformation is sound because $C[x]$ and x both have type o. We repeat this procedure until the term stabilizes. This algorithm clearly terminates since the size of the term decreases strictly after each step. The final term obtained is safe and of the same type as M. ☐

The previous argument crucially uses the fact that the type is generated from a single atom. It cannot be repeated for types generated from multiple atoms. In fact there are order-3 types with only 2 atoms that are inhabited by simply-typed terms but not by safe terms as example (i) below shows.

Example 3.75. Let a, b and c be three distinct atoms.

(i) Take the order-3 type $(((b,a),b),((a,b),a),a)$. Its normal inhabitants are given (up to α-conversion) by the following family of terms which are all unsafe:

$$\begin{array}{ll}
\lambda fg.g(\lambda x_1.f(\lambda y_1.x_1)) & \\
\lambda fg.g(\lambda x_1.f(\lambda y_1.g(\lambda x_2.y_1))) & \\
\lambda fg.g(\lambda x_1.f(\lambda y_1.g(\lambda x_2.f(\lambda y_2.x_i))) & \text{where } i = 1,2 \\
\lambda fg.g(\lambda x_1.f(\lambda y_1.g(\lambda x_2.f(\lambda y_2.g(\lambda x_3.y_i))) & \text{where } i = 1,2 \\
\ldots &
\end{array}$$

(ii) The order-3 type $(((a,c),b),((c,b),a),a)$ has for only normal inhabitant the unsafe term $\lambda fg.g(\lambda x.f(\lambda y.c))$.

(iii) For every $i,j,k \in \mathbb{N}$, let $\sigma(i,j,k)$ denote the type

$$\sigma(i,j,k) \equiv (i_a \to j_b) \to (j_b \to k_c) \to i_a \to k_c$$

where n_a denotes the type $(\ldots((a \to a) \to a)\ldots) \to a$ containing $n+1$ occurrences of a (as defined in Sec. 2.1.5). This type is inhabited by the "function composition term":

$$\lambda xyz.y(x\,z) \ .$$

[1]This problem was raised by Ugo dal Lago.

This term is safe if and only if $i \geq j$ (for the subterm $x\,z$ is safe iff $i = \operatorname{ord}(i_a) = \operatorname{ord} z \geq \operatorname{ord}(x\,z) = \operatorname{ord}(j_b) = j$). Therefore if $i \geq j$ the type $\sigma(i, j, k)$ is trivially inhabited by the above term. But there exist also values for i, j, k such that $i < j$ and $\sigma(i, j, k)$ is safely inhabited. For instance $\sigma(1, 3, 4)$ is inhabited by the safe term

$$\lambda x^{1_a \to 3_b} y^{3_b \to 4_c} z^{1_c}.y(x(\lambda u^a.u)) \ .$$

The order-4 type $\sigma(0, 2, 0)$, however, is unsafe: its only normal inhabitant is the unsafe term $\lambda xyz.y(xz)$.

(The first two examples are due to Luke Ong.)

3.5 Extensions

We now consider extensions of the safe simply-typed lambda calculus.

3.5.1 PCF

We define the language ***safe PCF*** as an applied version of the safe lambda calculus. Its types are the simple types over the single atomic type of natural numbers. It features the basic arithmetic operators of PCF (additions, substraction and conditional branching) as well as recursion. Equivalently, it is the restriction of PCF where the application and abstraction rules are constrained similarly as in the safe lambda calculus. The rules are given in Table 3.5. The circled rules are those that differ from their PCF counterpart.

Functional part

$$(\mathsf{var})\ \frac{}{\Gamma \vdash_{\mathsf{s}} x : A}\ x : A \in \Gamma \qquad (\mathsf{wk})\ \frac{\Gamma \vdash_{\mathsf{s}} M : A}{\Delta \vdash_{\mathsf{s}} M : A}\ \Gamma \subset \Delta \qquad (\delta)\ \frac{\Gamma \vdash_{\mathsf{s}} M : A}{\Gamma \Vdash_{\mathsf{app}} M : A}$$

$$(\mathsf{app_{as}})\ \frac{\Gamma \vdash_{\mathsf{s}} M : (A_1, \ldots, A_n, B) \quad \Gamma \vdash_{\mathsf{s}} N_1 : A_1 \quad \ldots \quad \Gamma \vdash_{\mathsf{s}} N_n : A_n}{\Gamma \Vdash_{\mathsf{app}} M\, N_1 \ldots N_n : B}$$

$$(\mathsf{app})\ \frac{\Gamma \vdash_{\mathsf{s}} M : (A_1, \ldots, A_n, B) \quad \Gamma \vdash_{\mathsf{s}} N_1 : A_1 \quad \ldots \quad \Gamma \vdash_{\mathsf{s}} N_n : A_n}{\Gamma \vdash_{\mathsf{s}} M\, N_1 \ldots N_n : B}\ \operatorname{ord} B \leq \operatorname{ord} \Gamma$$

$$(\mathsf{abs})\ \frac{\Gamma, x_1 : A_1, \ldots, x_n : A_n \Vdash_{\mathsf{app}} M : B}{\Gamma \vdash_{\mathsf{s}} \lambda x_1^{A_1} \ldots x_n^{A_n}.M : (A_1, \ldots, A_n, B)}\ \operatorname{ord}(A_1, \ldots, A_n, B) \leq \operatorname{ord} \Gamma$$

Arithmetic and recursion

$$(\mathsf{const})\ \frac{}{\vdash_{\mathsf{s}} n : \mathtt{exp}} \qquad (\mathsf{succ})\ \frac{\Gamma \vdash_{\mathsf{s}} M : \mathtt{exp}}{\Gamma \vdash_{\mathsf{s}} \mathtt{succ}\ M : \mathtt{exp}} \qquad (\mathsf{pred})\ \frac{\Gamma \vdash_{\mathsf{s}} M : \mathtt{exp}}{\Gamma \vdash_{\mathsf{s}} \mathtt{pred}\ M : \mathtt{exp}}$$

$$(\mathsf{cond})\ \frac{\Gamma \vdash_{\mathsf{s}} M : \mathtt{exp} \quad \Gamma \vdash_{\mathsf{s}} N_1 : \mathtt{exp} \quad \Gamma \vdash_{\mathsf{s}} N_2 : \mathtt{exp}}{\Gamma \vdash_{\mathsf{s}} \mathtt{cond}\ M\ N_1\ N_2} \qquad (\mathsf{rec})\ \frac{\Gamma \vdash_{\mathsf{s}} M : A \to A}{\Gamma \vdash_{\mathsf{s}} \mathsf{Y}_A M : A}$$

Table 3.5: Formation rules for safe PCF.

We extend the notion of *almost safety* (Sec. 3.1.4) to PCF: A PCF term is ***almost safe*** if it can be written $\lambda x_1 \ldots x_n.N_0 \ldots N_p$ for some $n, p \geq 0$ where N_i is safe for every $0 \leq i \leq p$.

Example 3.76. The addition function and equality test defined in Sec. 2.1.9 are typable in safe PCF.

The Substitution Lemma and No-variable-capture Lemma of the safe lambda calculus naturally extend to safe PCF. The small-step semantics of safe PCF is given by a relation $\rightarrow$ obtained from the one of PCF after substituting safe β-reduction (Def. 3.23) for β-reduction. The Subject Reduction Lemma from the safe lambda calculus implies that the relation $\rightarrow$ preserves safety: suppose that $M \rightarrow N$, then $\Gamma \vdash_{\mathsf{s}} M : T$ implies $\Gamma \vdash_{\mathsf{s}} N : T$. Similarly, the small-step reduction preserves almost-safety. Further it can again be proved that a term is safe if and only if its eta-long normal form is safe.

Remark concerning recursion

There are many ways to introduce recursion in the syntax of a programming language. In the presentation of PCF given in Sec. 2.1.9, recursion is introduced by mean of a set of constants Y_A, A ranging over PCF types, incarnating the Y combinator of the lambda calculus. The syntax is given by the rule (rec) of Table 3.5. For instance, the addition function can be represented by the PCF term:

$$\text{PLUS} \equiv \mathsf{Y}(\lambda p^{\mathtt{exp}\rightarrow\mathtt{exp}\rightarrow\mathtt{exp}} x^{\mathtt{exp}} y^{\mathtt{exp}}.\mathtt{cond}\ x\ y\ (p\ (\mathtt{pred}\ x)\ (\mathtt{succ}\ y))) \ .$$

Recursion can be introduced in different ways, however. For instance using the *least upper bound abstractor* 'μ' given by the formation rule

$$(\mu)\frac{\Gamma, f : A \vdash M : A}{\Gamma \vdash \mu f^A.M : A}$$

where the semantics of μ is given by the rule: $\mu f^A.M \rightarrow M[(\mu f^A.M)/f]$. Using this μ-construct, the addition function is defined as:

$$\text{PLUS} \equiv \mu p^{(\mathtt{exp}\rightarrow\mathtt{exp})\rightarrow\mathtt{exp}}.\lambda x^{\mathtt{exp}} y^{\mathtt{exp}}.\mathtt{cond}\ x\ y\ (p\ (\mathtt{pred}\ x)\ (\mathtt{succ}\ y)) \ .$$

Clearly in the context of PCF, these two definitions are interchangeable: $\mu f^A.M$ is equivalent to $\mathsf{Y}_A(\lambda f^A.M)$, and $\mathsf{Y}_A F$ is eta-equivalent to $\mathsf{Y}_A(\lambda f^A.F f)$ for some fresh variable f, which is equivalent to $\mu f^A.F f$.

In the context of safe PCF, however, the distinction is important. Indeed, let *safe μ-PCF* denote the calculus obtained by replacing the rule (rec) by (μ) in Table 3.5. Then we observe that safe PCF is strictly contained in safe μ-PCF. Indeed, compare the two ways of defining a recursive term:

$$\begin{array}{l} (\mathsf{abs}) \\ (\mathsf{rec}) \end{array} \dfrac{\dfrac{\Gamma, f : A \vdash_{\mathsf{s}} M : A}{\Gamma \vdash_{\mathsf{s}} \lambda f^A.M : A \rightarrow A}}{\Gamma \vdash_{\mathsf{s}} \mathsf{Y}_A(\lambda f^A.M)} \qquad\qquad (\mu)\, \frac{\Gamma, f : A \vdash_{\mathsf{s}} M : A}{\Gamma \vdash_{\mathsf{s}} \mu f^A.M : A}$$

Both derivations start with the premise $\Gamma, f : A \vdash_{\mathsf{s}} M : A$ which implies that $\operatorname{ord}\Gamma \geq \operatorname{ord} A$. But in the left derivation, before applying the Y combinator, we need first to abstract the variable f; this is done using the abstraction rule whose side-conditions gives $\operatorname{ord}\Gamma > \operatorname{ord} A$. The right derivation, however, only imposes the weaker condition $\operatorname{ord}\Gamma \geq \operatorname{ord} A$.

In fact, safe μ-PCF does not really deserve its name because the No-variable-capture lemma does not hold anymore in this language! Take for instance $\lambda f^{A\rightarrow B}\ a^A.(\lambda x^B.(\mu f^B.x))(f a)$ for every types A and B satisfying $\operatorname{ord} A \geq \operatorname{ord} B$. This term belongs to safe μ-PCF and it β-reduces to $\lambda f^{A\rightarrow B}\ a^A.(\mu f^B.x)[f a/x]$. But at this point it is not sound to push the substitution under the μ without first renaming the variables afresh as it would cause the variable f to be captured by μf.

Observe that if we were able to distinguish variables that are bound by λ from those bound by μ—for instance by tagging their occurrences appropriately—then the clash of variable names would be tolerable in this particular example since the two clashing occurrences of f are bound by a different kind of binder. Unfortunately, this argument cannot be generalized: there are safe μ-PCF terms that, when reduced using capture-permitting substitution, cause clashes between λ-bound variables. Take for instance:

$$M \equiv \lambda g^3\ h^3\ x^1.g(\mu F^3.N(F,g,h,x))$$
$$N(F,g,h,x) \equiv x(h(\lambda x^1.F(\lambda z^0.z)))$$

where 0 denotes the type o and $n+1$ denotes $n \rightarrow o$, for $n \in \mathbb{N}$. The safe μ-PCF term M reduces to:

$$\lambda g^3\ h^3\ x^1.g(x(h(\lambda \underline{x}^1.F(\lambda z^0.z))))[N(F,g,h,\underline{x})/F]\ ,$$

and performing this substitution *capture-permitting* would cause a clash between the two underlined variables.

The conclusion of this is that the definition that we really want for safe PCF is the one based on the Y combinator. Another reason why safe μ-PCF is not an interesting language is that the game-semantic characterization of safe PCF that we will establish in Chapter 6 does not hold in safe μ-PCF.

3.5.1.1 Expressivity

In the lambda calculus, the safety condition significantly limits the expressivity of the language: as we have observed before, the conditional function over Church numerals is for instance not definable in the safe lambda calculus. On the other hand in safe PCF the conditional operator comes for free since the arithmetic constructs are built in the language. So the question is: *Does safety genuinely restrict the power of PCF?* We first show that safe PCF is a non-trivial language by proving, using a reduction from the QUEUE-HALTING problem, that the termination problem is not decidable. We further observe that despite the strong constraint imposed by safety, the presence of recursion gives back to safe PCF the computational power of a full-fledged Turing complete language.

The Queue programming system We fix a finite alphabet $\Sigma = \{a_1, \ldots, a_p\}$. A QUEUE program is a finite sequence of instructions that manipulate a FIFO (First In First Out) queue data-structure. A program P is a sequence of n instructions for some $n \in \mathbb{N}$. For $1 \leq i \leq n$ we write $P.i$ to denote the i^{th} instruction of P. There are four kinds of instructions: halting, enqueuing, dequeuing and branching. The set of instructions is given by:

$$\mathcal{I} = \{\texttt{halt}\} \cup \{\texttt{enqueue}\ a \mid a \in \Sigma\} \cup \{\texttt{dequeue}\} \cup \{\texttt{goto}\ l\ \texttt{if first} = a \mid l \in 1..n, a \in \Sigma\}\ .$$

The operational semantics is described using a set of states $\{\mathsf{halted}\} \cup \{1,..,n\} \times \Sigma^*$. The special state halted is the end-of-program state that is reached when the program terminates. A state of the form $(i,x) \in \{1,..,n\} \times \Sigma^*$ indicates that the queue's content is given by the sequence x and that the next instruction to be executed by the machine is $P.i$. The empty queue is represented by the empty sequence ϵ, and for every sequence $x \in \Sigma^*$, the first element of x corresponds to the element that has been *first* enqueued (*i.e.*, the queue is fed at the right-end side and consumed at the left-end side). The operational semantics is defined by the following rules:

$$\begin{array}{rcl}
(i,x) \text{ with } P.i = \texttt{halt} & \rightarrow & \mathsf{halted} \\
(i,x) \text{ with } P.i = \texttt{enqueue}\ a & \rightarrow & (i+1, x \cdot a) \\
(i,\epsilon) \text{ with } P.i = \texttt{dequeue} & \rightarrow & \mathsf{halted} \\
(i,a \cdot x) \text{ with } P.i = \texttt{dequeue} & \rightarrow & (i+1, x) \\
(i,\epsilon) \text{ with } P.i = \texttt{goto}\ l\ \texttt{if first} = a & \rightarrow & (i+1, \epsilon) \\
(i,b \cdot x) \text{ with } a \neq b \text{ and } P.i = \texttt{goto}\ l\ \texttt{if first} = a & \rightarrow & (i+1, b \cdot x)
\end{array}$$

$$(i, a\cdot x) \text{ with } P.i = \texttt{goto}\ l\ \texttt{if first} = a \quad \rightarrow \quad (l, a\cdot x)\ .$$

We write $\rightarrow^*$ to denote the reflexive transitive closure of $\rightarrow$.

The QUEUE-HALTING problem ("Given a QUEUE program, will it halt eventually?") is undecidable. This is because Post's Tag Systems, which are Turing complete [CM64], can be simulated [Min67] in QUEUE.

Encoding Queue-Halting in safe PCF Given a QUEUE program P with n instructions, we construct a safe PCF term $\vdash_{\mathsf{s}} M_P : \texttt{exp}$ that simulates P in the sense that $P \Downarrow$ if and only if $M_P \rightarrow^* \mathsf{halted}$.

Queue encoding: We fix a distinguished element $\perp$ denoting the end of the queue. Let $\Sigma^{\perp} = \Sigma \cup \{\perp\}$. We identify each queue content $s \in \Sigma^*$ with the infinite sequence $s\perp^{\omega} \in \Sigma^{\omega}$. We assume that an injective encoding function $\Sigma^{\perp} \longrightarrow \mathbb{N}$ is given and we write $\overline{a}$ to denote the encoding of an element in $\Sigma^{\perp}$. (For instance take $\overline{\perp} = 0$ and $\overline{a_k} = k$ for $1 \leq k \leq p$.)

We say that a *PCF* term M *computes* the queue content s if and only if $M\,k \Downarrow \overline{s_k}$ for every $k \in \mathbb{N}$. For every queue-content $s \in \Sigma^*$ we define the safe PCF term

$$\vdash_{\mathsf{s}} \overline{s} \equiv \lambda i^{\texttt{exp}}.\texttt{match}\, i\, \texttt{with}\, 0 \rightarrow \overline{s_0} \mid \ldots \mid n \rightarrow \overline{s_{|s|-1}} \mid \_ \rightarrow \overline{\perp} : \texttt{exp} \rightarrow \texttt{exp}$$

which clearly computes s. The length $|s|$ of the queue can then by computed by the term

$$\begin{aligned}\vdash_{\mathsf{s}} \text{LENGTH} \equiv Y(\lambda f^{\texttt{exp}\rightarrow(\texttt{exp}\rightarrow\texttt{exp})\rightarrow\texttt{exp}}\, k^{\texttt{exp}}\, x^{\texttt{exp}\rightarrow\texttt{exp}}\ .\\ \texttt{if}\, x\,k = \overline{\perp} \texttt{then}\, k\ \texttt{else}\, f\,(k+1)\,x)\ 0 : (\texttt{exp} \rightarrow \texttt{exp}) \rightarrow \texttt{exp}\end{aligned}$$

satisfying $\text{LENGTH}\ \overline{s} \Downarrow |s|$ for all $s \in \Sigma^*$.

Instruction encoding: We assume an injective function $\mathcal{I} \rightarrow \mathbb{N}$ encoding each instruction c of $\mathcal{I}$ as a natural number $\overline{c}$. An example is the following function defined for $1 \leq i \leq p, 1 \leq l \leq n$:

$c \in \mathcal{I}$	halt	dequeue	enqueue a_i	goto l if first= a_i
$\overline{c} \in \mathbb{N}$	0	1	$1+i$	$1+p+n.l+i$

A QUEUE program P is then compiled to the safe PCF term:

$$\vdash_{\mathsf{s}} \overline{P} \equiv \lambda i^{\texttt{exp}}.\texttt{match}\, i\, \texttt{with}\, 0 \rightarrow \overline{P.0} \mid \ldots \mid n \rightarrow \overline{P.n} \mid \_ \rightarrow \overline{\texttt{halt}} : \texttt{exp} \rightarrow \texttt{exp}$$

so that for all $i \in \mathbb{N}$, $\overline{P}\,i$ evaluates to the encoding of the i^{th} instruction of P. We can now define an interpreter SIM_P for QUEUE-programs given in compiled form $\overline{P}$:

$$\begin{array}{ll} \multicolumn{2}{l}{\vdash_{\mathsf{s}} \text{SIM}_P \equiv Y(\lambda f^{(\texttt{exp},(\texttt{exp},\texttt{exp}),\texttt{exp})}\ i^{\texttt{exp}}\ x^{(\texttt{exp},\texttt{exp})}.} \\ \multicolumn{2}{l}{\quad \texttt{match}\ \overline{P}\,i\ \texttt{with}} \\ \quad\ \overline{\texttt{halt}} & \rightarrow\ 0 \\ \quad \mid \overline{\texttt{dequeue}} & \rightarrow\ f(i+1)(\lambda j^{\texttt{exp}}.x(j+1)) \\ \quad \mid \overline{\texttt{enqueue}\ a_1} & \rightarrow\ f(i+1)(\lambda j^{\texttt{exp}}.\texttt{if}\ j = \text{LENGTH}\ x\,\texttt{then}\,\overline{a_1}\ \texttt{else}\,x\,j) \\ \quad \ldots & \\ \quad \mid \overline{\texttt{enqueue}\ a_p} & \rightarrow\ f(i+1)(\lambda j^{\texttt{exp}}.\texttt{if}\ j = \text{LENGTH}\ x\,\texttt{then}\,\overline{a_p}\ \texttt{else}\,x\,j) \\ \quad \mid \overline{\texttt{goto}\ l\ \texttt{if first} = a_1} & \rightarrow\ \texttt{if}\ \text{LENGTH}\ x = 0\ \texttt{then}\, f\,(i+1)\,x \\ & \quad\ \texttt{else if}\ \overline{a_1} = x\,0\ \texttt{then}\ f\,l\,x \\ & \quad\ \texttt{else}\ \ f\,(i+1)\,x \\ \quad \ldots & \\ \quad \mid \overline{\texttt{goto}\ l\ \texttt{if first} = a_p} & \rightarrow\ \texttt{if}\ \text{LENGTH}\ x = 0\ \texttt{then}\, f\,(i+1)\,x \\ & \quad\ \texttt{else if}\ \overline{a_p} = x\,0\ \texttt{then}\ f\,l\,x \\ & \quad\ \texttt{else}\, f\,(i+1)\,x \\ \multicolumn{2}{l}{\quad)\ 0\ \overline{\epsilon} : \texttt{exp}\ .} \end{array}$$

Clearly the term SIM_P is safe and simulates the QUEUE program P in the sense that $\text{SIM}_P \Downarrow$ if and only if $P \rightarrow^* \mathsf{halted}$. Hence

Theorem 3.77. *The* HALTING *problem for (the 2nd order fragment of) safe PCF is undecidable.*

Since the HALTING is reducible to the observational equivalence problem, this also implies that observational equivalence for the 2nd-order fragment of safe PCF (with Y_1 recursion and unbounded base types) is undecidable. This result is not surprising: it is easy to see that the partial recursive functions are computable in the order 2 fragment of safe PCF, and hence safe PCF is Turing complete. (This can also be proved by simulating Turing machines in safe PCF using an encoding similar to the one used above.)

The reason why these encodings work is because unsafety only appears at order 3 in PCF, and the 2nd order fragment of PCF is already Turing complete.

Loader has shown [Loa01] that observational equivalence for *finitary* PCF (the fragment with no recursion and finite base types) is already undecidable at order 5. It is unknown whether this result still holds for finitary safe PCF.

3.5.2 Idealized Algol

In this section we present two possible approaches to extend the safety restriction to a language featuring block-variable constructs such as Idealized Algol. This gives rise to two different versions of "Safe Idealized Algol". In the first version, all free variables are required to satisfy the safety constraint whereas in the second version, variables declared with a block-allocated construct are not required to satisfy the safety constraint. We then show that the good properties of the safe lambda calculus remain in these two extensions of the safe lambda calculus.

3.5.2.1 Strongly Safe IA

The most immediate way to introduce the safety constraint for IA terms consists in adding the typing rules for IA constants to the typing system of the safe lambda calculus. Equivalently, this means taking the system of rules of IA and replacing the application and abstraction rules by those of the safe lambda calculus. We refer to this language as ***strongly safe IA***. The rules are formally given in Table 3.7. The rules circled in the table are those that differ from their IA counterpart.

This language satisfies the basic property of the safe lambda calculus: Free variables have order greater or equal to the order of the term. It is interesting to note that the typing rules of IA do not need to be modified for this property to hold. In particular, the rule (new) allows one to "abstract" variables without having to satisfy any side-condition, contrary to the lambda-abstraction rule (abs). Such side-condition is unnecessary because the block-allocation construct produces a term with the same type as the term in the premise of the rule. Therefore the basic property trivially holds.

On the other hand, this ability to "abstract" variables without increasing the order of the term as a downside: the No-variable-capture result—that it is no necessary to rename variables afresh when performing substitution—does not hold anymore, at least in its original formulation. Take for instance the following strongly-safe term-in-context:

$$x : \mathtt{var} \vdash_{ss} (\lambda y^{\mathtt{exp}}.\mathtt{new}\ x\ \mathtt{in}\ y)(\mathtt{deref}\ x) \equiv M_1 : \mathtt{exp}\ .$$

Then we have:

$$M_1 \to_\beta (\mathtt{new}\ x\ \mathtt{in}\ y)\,[(\mathtt{deref}\ x)/y]\ .$$

Performing the substitution without renaming variables afresh causes the variable x to get captured by the innermost $\mathtt{new}\ x$ giving $\mathtt{new}\ x\ \mathtt{in}\ \mathtt{deref}\ x$. On the other hand the standard substitution gives $\mathtt{new}\ z\ \mathtt{in}\ \mathtt{deref}\ x$. These two terms are clearly not observationally equivalent. Conclusion: it is *not* "safe" to use capture-permitting substitution on strongly-safe IA terms!

A weaker version of the No-variable-capture lemma can be stated though. We can defined an alternative notion of capture-permitting substitution, called *semi-capture permitting substitution*, that behaves like the usual capture-permitting substitution except that it renames

Functional part

$$(\mathsf{var})\ \frac{}{\Gamma \vdash_{ss} x : A}\quad x : A \in \Gamma \qquad (\mathsf{wk})\ \frac{\Gamma \vdash_{ss} M : A}{\Delta \vdash_{ss} M : A}\quad \Gamma \subset \Delta \qquad (\delta)\ \frac{\Gamma \vdash_{ss} M : A}{\Gamma \Vdash_{\mathsf{app}} M : A}$$

$$(\mathsf{app_{as}})\ \frac{\Gamma \vdash_{ss} M : (A_1, \ldots, A_n, B) \quad \Gamma \vdash_{ss} N_1 : A_1 \quad \ldots \quad \Gamma \vdash_{ss} N_n : A_n}{\Gamma \Vdash_{\mathsf{app}} M\ N_1 \ldots N_n : B}$$

$$(\mathsf{app})\ \frac{\Gamma \vdash_{ss} M : (A_1, \ldots, A_n, B) \quad \Gamma \vdash_{ss} N_1 : A_1 \quad \ldots \quad \Gamma \vdash_{ss} N_n : A_n}{\Gamma \vdash_{\mathsf{s}} M\ N_1 \ldots N_n : B}\quad \mathrm{ord}\, B \leq \mathrm{ord}\, \Gamma$$

$$(\mathsf{abs})\ \frac{\Gamma, x_1 : A_1, \ldots, x_n : A_n \Vdash_{\mathsf{app}} M : B}{\Gamma \vdash_{ss} \lambda x_1^{A_1} \ldots x_n^{A_n}.M : (A_1, \ldots, A_n, B)}\quad \mathrm{ord}\,(A_1, \ldots, A_n, B) \leq \mathrm{ord}\, \Gamma$$

Arithmetic and recursion

$$(\mathsf{const})\ \frac{}{\vdash_{ss} n : \mathtt{exp}} \qquad (\mathsf{succ})\ \frac{\Gamma \vdash_{ss} M : \mathtt{exp}}{\Gamma \vdash_{ss} \mathtt{succ}\ M : \mathtt{exp}} \qquad (\mathsf{pred})\ \frac{\Gamma \vdash_{ss} M : \mathtt{exp}}{\Gamma \vdash_{ss} \mathtt{pred}\ M : \mathtt{exp}}$$

$$(\mathsf{cond})\ \frac{\Gamma \vdash_{ss} M : \mathtt{exp} \quad \Gamma \vdash_{ss} N_1 : \mathtt{exp} \quad \Gamma \vdash_{ss} N_2 : \mathtt{exp}}{\Gamma \vdash_{ss} \mathtt{cond}\ M\ N_1\ N_2} \qquad (\mathsf{rec})\ \frac{\Gamma \vdash_{ss} M : A \to A}{\Gamma \vdash_{ss} \mathsf{Y}_A M : A}$$

Imperative constructs

$$(\mathsf{seq})\ \frac{\Gamma \vdash_{ss} M : \mathtt{com} \quad \Gamma \vdash_{ss} N : A}{\Gamma \vdash_{ss} \mathtt{seq}_A\ M\ N\ : A}\quad A \in \{\mathtt{com}, \mathtt{exp}\}$$

$$(\mathsf{assign})\ \frac{\Gamma \vdash_{ss} M : \mathtt{var} \quad \Gamma \vdash_{ss} N : \mathtt{exp}}{\Gamma \vdash_{ss} \mathtt{assign}\ M\ N\ : \mathtt{com}} \qquad (\mathsf{deref})\ \frac{\Gamma \vdash_{ss} M : \mathtt{var}}{\Gamma \vdash_{ss} \mathtt{deref}\ M\ : \mathtt{exp}}$$

$$(\mathsf{new})\ \frac{\Gamma, x : \mathtt{var}\ \vdash_{ss} M : A}{\Gamma \vdash_{ss} \mathtt{new}\ x\ \mathtt{in}\ M : A}\quad A \in \{\mathtt{com}, \mathtt{exp}\}$$

$$(\mathsf{mkvar})\ \frac{\Gamma \vdash_{ss} M_1 : \mathtt{exp} \to \mathtt{com} \quad \Gamma \vdash_{ss} M_2 : \mathtt{exp}}{\Gamma \vdash_{ss} \mathtt{mkvar}\ M_1\ M_2\ : \mathtt{var}}$$

Table 3.6: Formation rules for strongly safe IA.

block-allocated variables afresh upon performing substitution. The No-variable-capture lemma for strongly safe IA then becomes: "Substitution can be safely implemented by semi-capture permitting substitution".

3.5.2.2 Safe IA

It turns out that the definition of strongly safe IA is too restrictive and we can identify a larger fragment in which the so-called "No-variable-capture" lemma holds. Consider the following IA term:

$$\vdash \mathtt{new}\ x\ \mathtt{in}\ \underline{(\lambda z^{\mathtt{exp}}.\mathtt{deref}\,\underline{x})}\,0 : \mathtt{exp}\ .$$

It is not strongly safe since the variables $x : \mathtt{var}$ and $z : \mathtt{exp}$ have the same order but are not abstracted together. However x is a block-allocated variable so no term can ever be substituted for such variable when performing reduction: morally this term should be considered safe. We thus observe that there is no gain in constraining occurrences of block-allocated variables.

We will therefore distinguish two kinds of variables in a closed term: the "standard ones"—those that are bound by λ-abstractions—and the "imperative" ones—those that are declared by a block-allocation construct—and we will change the side-condition of the abstraction rule so that only variables of the first kind are constrained.

It is also possible to relax the safety constraint for another class of variables. Among the lambda-bound variables, we consider the subclass of variables that are bound by a lambda node $\lambda x^{\mathtt{exp}}$ inside a term of the form $\mathtt{mkvar}(\lambda x^{\mathtt{exp}}.M)N$. We call these variables `mkvar`-*bound variables*. It turns out that it is also possible to relax the safety constraint for this class of variables. To see why this is the case, we need to redefine the typing rules for the `mkvar` construct: we replace the typing in two steps (first abstracting x in M and then constructing $\mathtt{mkvar}(\lambda x^{\mathtt{exp}}.M)N$) by a single typing rule forming $\mathtt{mkvar}(\lambda x^{\mathtt{exp}}.M)N$ directly from M and N. These two ways of typing the `mkvar` construct are semantically equivalent because it is always possible to eta-expand the first argument of `mkvar` into a term of the form $\lambda x^{\mathtt{exp}}.M$.

The small step semantics is then redefined by replacing the rule

$$\mathtt{assign}\ (\mathtt{mkvar} MN)\ n \rightarrow Mn$$

by

$$\mathtt{assign}\ (\mathtt{mkvar}(\lambda x^{\mathtt{exp}}.M)N)\ n \rightarrow M\,[n/x]\ . \tag{3.4}$$

This change highlights the fact that no substitution can ever occur for the variable x in the term M, there is therefore no need to enforce the safety constraint for free occurrences of the variable x in M.

These remarks lead us a more general notion of safety for IA. We consider new judgements of the form $\Gamma|\Xi \vdash_s M : A$, called ***split terms-in-context***[2], where the context is partitioned into two *disjoint* components: The first component Γ contains the lambda-bound variables that are constrained by the safety restriction; the second component contains block-declared variables as well as `mkvar`-bound variables. The component Ξ contains variables of type `var` and `exp` only, while the other component may contain variables of any type including `var`. It is straightforward to redefine the typing rule of IA in such a way that these two distinct contexts are maintained appropriately. In particular:

(i) The abstraction rules can only abstract variables from the first component of the context;

(ii) The `new` and `mkvar` constructs can only bind variables from the second context component;

(iii) The side-condition in the abstraction rules constrains only variables from the first context component.

[2] This terminology is borrowed from Abramsky and McCusker's tutorial on game semantics [AM98b]

The typing system for this new judgement is given in Table 3.7; the circled rules highlight the important changes from the rules of Table 3.6. A split-term with an empty context Ξ is called a ***semi-closed split-term***. We define ***safe IA*** to be the set of *semi-closed* split-terms typable with the system of rules of Table 3.7. For convenience we introduce the additional rule

$$\frac{\Gamma \,|\, \emptyset \vdash_s M : A}{\Gamma \vdash_s M : A}$$

so that safe IA is equivalently given by the set of terms-in-context $\Gamma \vdash_s M : A$.

Example 3.78. Strongly safe IA is a subset of safe IA. The following example shows that the inclusion is strict:

$$\vdash_s \lambda f^{(\mathtt{exp}\to\mathtt{com})\to\mathtt{exp}}.\ \mathtt{new}\ i\ \mathtt{in}\ f(\lambda x^{\mathtt{exp}}.\mathtt{assign}\ i\ x) : \mathtt{exp}$$

$$\text{but } \not\vdash_{ss} \lambda f^{(\mathtt{exp}\to\mathtt{com})\to\mathtt{exp}}.\ \mathtt{new}\ i\ \mathtt{in}\ f(\lambda x^{\mathtt{exp}}.\mathtt{assign}\ i\ x) : \mathtt{exp}\ .$$

It is not strongly safe because the variables i and x are of the same order but only x is abstracted by the lambda. It is safe because unsafe occurrences of block-allocated variables such as i are tolerated in safe IA.

Example 3.79. The following term is a safe IA beta-normal term:

$$f : ((\mathtt{exp} \to \mathtt{exp}) \to \mathtt{com}) \vdash_{\mathsf{s}} \mathtt{mkvar}\ (\lambda x^{\mathtt{exp}}.f(\lambda y^{\mathtt{exp}}.\underline{x}))\ 0 : \mathtt{com}^{\omega} \times \mathtt{exp}\ .$$

Observe that the unsafe occurrence of the variable x is tolerated because it is a `mkvar`-bound variable.

Since in split safe IA terms, only the variables from the left context component are constrained by the safety restriction, thus the basic property of the safe lambda calculus (Lemma 3.8) becomes:

Lemma 3.80. *Suppose* $\Gamma|\Xi \vdash_s M : A$. *Then*

$$\forall z : A \in \Gamma . z \in FV(M) \implies \operatorname{ord} z \geq \operatorname{ord} A\ .$$

The small-step reduction semantics of safe IA is defined similarly as in Sec. 2.1.10 except that β-reduction is replaced by safe β-reduction and the rules for `mkvar` are redefined according to (3.4). Again it is easy to see that safety is preserved by the small-step reduction of IA:

Lemma 3.81 (Reduction preserves safety)**.** *Let M be an IA term and $\rightarrow$ denotes the small-step reduction of safe IA. Then* $\Gamma\ |\Xi \vdash_{\mathsf{s}} M : A \wedge M \to N \implies \Gamma\ |\Xi \vdash_{\mathsf{s}} N : A\ .$

The proof is by an easy induction.

3.5.2.3 No-variable capture lemma

In which sense are the two calculi above-defined "safe"? In the lambda calculus fragment, the term "safe" refers to the fact that under the *safe typing convention*, substitution can be performed *capture-permitting*. Unfortunately, as we have observed before, in the presence of block-allocation constructs this lemma does not hold anymore because the block-allocation construct `new` does not increase the order of the term that is being formed contrary to λ-abstractions—a property that is crucially used in the proof of the No-variable-capture lemma. The following examples illustrate this. Consider the terms:

$$M_1 \equiv \mathtt{new}\ x\ \mathtt{in}\ \mathtt{seq}\ (\mathtt{assign}\ x\ 1)\ ((\lambda y^0.\mathtt{new}\ x\ \mathtt{in}\ y)(\mathtt{deref}\ x))$$
$$M_2 \equiv \lambda x^1.(\lambda y^1.(\mathtt{new}\ x\ \mathtt{in}\ y\ 0))x$$

Functional part

$$(\mathsf{var}^{\mathtt{var}})\ \frac{}{\emptyset|\Xi \vdash_s x : \mathtt{var}}\ x : \mathtt{var} \in \Xi \qquad (\mathsf{var}^{\mathtt{exp}})\ \frac{}{\emptyset|\Xi \vdash_s x : \mathtt{exp}}\ x : \mathtt{exp} \in \Xi$$

$$(\mathsf{var})\ \frac{}{\Gamma|\emptyset \vdash_s x : A}\ x : A \in \Gamma \qquad (\mathsf{wk})\ \frac{\Gamma|\Xi \vdash_s M : A}{\Gamma'|\Xi \vdash_s M : A}\ \Gamma \subset \Gamma' \ \wedge\ \mathrm{dom}(\Gamma') \cap \mathrm{dom}(\Xi) = \emptyset$$

$$(\delta)\ \frac{\Gamma \vdash_{\mathsf{s}} M : A}{\Gamma \Vdash_{\mathsf{app}} M : A} \qquad (\mathsf{app}_{\mathsf{as}})\ \frac{\Gamma|\Xi \vdash_s M : (A_1, \ldots, A_n, B) \quad \Gamma|\Xi \vdash_s N_1 : A_1 \ \ldots\ \Gamma|\Xi \vdash_s N_n : A_n}{\Gamma|\Xi \Vdash_{\mathsf{app}} M N_1 \ldots N_n : B}$$

$$(\mathsf{app})\ \frac{\Gamma|\Xi \vdash_s M : (A_1, \ldots, A_n, B) \quad \Gamma|\Xi \vdash_s N_1 : A_1 \quad \ldots \quad \Gamma|\Xi \vdash_s N_n : A_n}{\Gamma|\Xi \vdash_s M N_1 \ldots N_n : B}\ \mathrm{ord}\, B \leq \mathrm{ord}\, \Gamma$$

$$(\mathsf{abs})\ \frac{\Gamma, x_1 : A_1, \ldots, x_n : A_n|\Xi \Vdash_{\mathsf{app}} M : B}{\Gamma|\Xi \vdash_s \lambda x_1^{A_1} \ldots x_n^{A_n}.M : (A_1, \ldots, A_n, B)}\ \mathrm{ord}\,(A_1, \ldots, A_n, B) \leq \mathrm{ord}\, \Gamma$$

Arithmetic and recursion

$$(\mathsf{const})\ \frac{}{\emptyset|\emptyset \vdash_s n : \mathtt{exp}} \qquad (\mathsf{succ})\ \frac{\Gamma|\Xi \vdash_s M : \mathtt{exp}}{\Gamma|\Xi \vdash_s \mathtt{succ}\ M : \mathtt{exp}} \qquad (\mathsf{pred})\ \frac{\Gamma|\Xi \vdash_s M : \mathtt{exp}}{\Gamma|\Xi \vdash_s \mathtt{pred}\ M : \mathtt{exp}}$$

$$(\mathsf{cond})\ \frac{\Gamma|\Xi \vdash_s M : \mathtt{exp} \quad \Gamma|\Xi \vdash_s N_1 : \mathtt{exp} \quad \Gamma|\Xi \vdash_s N_2 : \mathtt{exp}}{\Gamma|\Xi \vdash_s \mathtt{cond}\ M\ N_1\ N_2} \qquad (\mathsf{rec})\ \frac{\Gamma|\Xi \vdash_s M : A \to A}{\Gamma|\Xi \vdash_s \mathsf{Y}_A M : A}$$

Imperative constructs

$$(\mathsf{seq})\ \frac{\Gamma|\Xi \vdash_s M : \mathtt{com} \quad \Gamma|\Xi \vdash_s N : A}{\Gamma|\Xi \vdash_s \mathtt{seq}_A\ M\ N\ : A}\quad A \in \{\mathtt{com}, \mathtt{exp}\}$$

$$(\mathsf{assign})\ \frac{\Gamma|\Xi \vdash_s M : \mathtt{var} \quad \Gamma|\Xi \vdash_s N : \mathtt{exp}}{\Gamma|\Xi \vdash_s \mathtt{assign}\ M\ N\ : \mathtt{com}} \qquad (\mathsf{deref})\ \frac{\Gamma|\Xi \vdash_s M : \mathtt{var}}{\Gamma|\Xi \vdash_s \mathtt{deref}\ M\ : \mathtt{exp}}$$

$$(\mathsf{new})\ \frac{\Gamma|\Xi, x : \mathtt{var} \ \vdash_s M : A}{\Gamma|\Xi \vdash_s \mathtt{new}\ x\ \mathtt{in}\ \ M : A}\quad A \in \{\mathtt{com}, \mathtt{exp}\}$$

$$(\mathsf{mkvar})\ \frac{\Gamma|\Xi, x : \mathtt{exp} \vdash_s M_1 : \mathtt{exp} \to \mathtt{com} \quad \Gamma|\Xi \vdash_s M_2 : \mathtt{exp}}{\Gamma|\Xi \vdash_s \mathtt{mkvar}\ (\lambda x^{\mathtt{exp}}.M_1)\ M_2\ : \mathtt{var}}\quad 1 \leq \mathrm{ord}\, \Gamma$$

Table 3.7: Formation rules for safe IA.

$$M_3 \equiv \lambda f^2.\mathtt{new}\ x\ \mathtt{in}\ (\lambda y^1.f(\lambda x^0.y))(\lambda z^0.\mathtt{deref}\ \underline{x})$$
$$M_4 \equiv \lambda x^{\mathtt{com}}.(\lambda y^{\mathtt{com}}.\mathtt{mkvar}\,(\lambda x^{\mathtt{exp}}.y)\,0)x$$

where the type n, for $n \in \mathbb{N}$, is an abbreviation for $n_{\mathtt{exp}}$.

All these terms are safe IA terms (but only M_1 and M_2 are strongly safe) and contracting the redexes in those terms using capture-permitting substitution causes problematic variable captures:

(i) For M_1, performing the substitution without renaming variables afresh causes the capture of x by the innermost $\mathtt{new}\ x$, giving $\mathtt{new}\ x\ \mathtt{in}\ \mathtt{seq}\ (\mathtt{assign}\ x\ 1)\ (\mathtt{new}\ x\ \mathtt{in}\ \mathtt{deref}\ x)$ which is observationally equivalent to 0 (since block-allocated variables are initialized with 0). On the other hand standard substitution gives $\mathtt{new}\ x\ \mathtt{in}\ \mathtt{seq}\ (\mathtt{assign}\ x\ 1)\ (\mathtt{new}\ z\ \mathtt{in}\ \mathtt{deref}\ x)$ which is observationally equivalent to 1.

(ii) For M_2, the capture-permitting substitution gives $\lambda x^1.(\mathtt{new}\ x\ \mathtt{in}\ x\ 0)$ which is not even typable in IA;

(iii) For M_3, capture-permitting substitution gives $\lambda f^2.\mathtt{new}\ x\ \mathtt{in}\ \lambda y^1.f(\lambda x^0.(\lambda x^0.\mathtt{deref}\,x))$ which is not a typable IA term;

(iv) Finally for M_4, capture-permitting substitution gives $(\lambda y^{\mathtt{com}}.\mathtt{mkvar}\,(\lambda x^{\mathtt{exp}}.x)\,0)$ which is not a typable IA term because the subterm $\lambda x^{\mathtt{exp}}.x$ is of type $\mathtt{exp} \rightarrow \mathtt{exp}$ instead of the required type $\mathtt{exp} \rightarrow \mathtt{com}$.

To deal with the first two examples, we have no other choice than renaming block-declared variables afresh upon substitution. For the last two kinds of variable capture (which only happen for safe terms that are not strongly safe) we can resolve the problem by adopting the following convention:

Convention 3.82 The set of names used for block-declared and mkvar-bound variables is disjoint from the set of names used for lambda-abstracted variables. This convention can be enforced by tagging each variable occurrence to indicate whether it is a block-allocated variable or a lambda-abstracted variable, thus permitting one to resolve any binding ambiguity. Observe that this convention is stronger than requiring that the sets of names of the two context components of a split-term are disjoint because the latter only constrains the free variables of the term whereas what we are requiring here is a global constraint on all variable names occurring in the term including the bound ones.

This leads us to the following notion of substitution which performs variable renaming only for block-allocated variables and mkvar-bound variables:

Definition 3.83. The ***semi-capture-permitting substitution*** of the term-in-context $\Gamma|\Xi \vdash N : A$ for x in the term-in-context $\Gamma, x : A|\Xi \vdash M : B$ is given by $\Gamma|\Xi \vdash M\{\!\{N/x\}\!\}$ where the operation $\{\!\{N/x\}\!\}$ is defined inductively on M as follows:

$$\begin{aligned}
x\{\!\{N/x\}\!\} &= N \\
y\{\!\{N/x\}\!\} &= y && y \neq x; \\
(\lambda x^\tau.M)\{\!\{N/x\}\!\} &= \lambda x^\tau.M \\
(\lambda y^\tau.M)\{\!\{N/x\}\!\} &= \lambda y^\tau.M\{\!\{N/x\}\!\} && \text{where } y \neq x; \\
(\mathtt{new}\ x\ \mathtt{in}\ M)\{\!\{N/x\}\!\} &= \mathtt{new}\ x\ \mathtt{in}\ M \\
(\mathtt{new}\ y\ \mathtt{in}\ M)\{\!\{N/x\}\!\} &= \mathtt{new}\ z\ \mathtt{in}\ M\{\!\{z/y\}\!\}\{\!\{N/x\}\!\} && \text{if } x \neq y,\ z \text{ fresh}; \\
(\mathtt{mkvar}\,(\lambda x^\tau.M_1)\,M_2)\{\!\{N/x\}\!\} &= \mathtt{mkvar}\,(\lambda x^\tau.M_1)\,M_2\{\!\{N/x\}\!\} \\
(\mathtt{mkvar}\,(\lambda y^\tau.M_1)\,M_2)\{\!\{N/x\}\!\} &= \mathtt{mkvar}\,(\lambda z^\tau.M_1\{\!\{z/y\}\!\}\{\!\{N/x\}\!\})\,M_2\{\!\{N/x\}\!\} && \text{if } x \neq y,\ z \text{ fresh}.
\end{aligned}$$

The other constants and application cases are defined inductively in the standard way.

It is now possible to state a version of the *No-variable capture lemma* for safe IA:

Lemma 3.84 (No-variable capture). *Suppose that* $\Gamma|\Xi \vdash_s N : A$ *and* $\Gamma, x : A|\Xi \vdash_s M : B$*. Then the substitution* $M[N/x]$ *can be performed semi-capture-permitting:*

$$M[N/x] \equiv M\{\!\![N/x]\!\!\} \ ,$$

provided that either

(i) convention 3.13 and 3.82 are taken;

(ii) or convention 3.82 is taken and $\Gamma|\Xi \vdash M\{N/x\} : B$ *is a valid (not-necessarily safe) IA judgement.*

The proof is a trivial extension of Lemma 3.15 and 3.17.

Corollary 3.85. *Let* $\Gamma \vdash_s N : A$ *and* $\Gamma, x : A \vdash_s M : B$ *be safe IA terms-in-context.*

(i) If convention 3.13 is adopted then $M[N/x] \equiv M\{\!\![N/x]\!\!\}$*;*

(ii) If $\Gamma \vdash M\{N/x\} : B$ *is typable in IA then* $M[N/x] \equiv M\{\!\![N/x]\!\!\}$*.*

3.5.3 Generalization to other applied lambda calculi

In this section, we define the notion of safety for every given applied lambda calculus extended with a stock of interpreted constants Σ but without recursion. The syntax of the language is given by some system of rules producing split-terms of the form

$$\Gamma \,|\, \Xi \vdash M : T$$

for some simple-type T, where variables in the context Γ and Ξ are called the Γ-variables and Ξ-variables respectively. The calculus must satisfy the following prerequisites:
(i) The abstraction rule can only abstract Γ-variables;
(ii) The terms of the languages are given by the semi-closed split-terms $\Gamma|\emptyset \vdash M : T$ abbreviated as $\Gamma \vdash M : T$.
Consequently, a Ξ-variable can only be "bound" by some constant construct of the language but not by a lambda-abstraction.

Definition 3.86. Consider an applied lambda calculus as defined above. Its ***safe fragment*** is defined as the system obtained by restricting the pure lambda calculus fragment of the language in such a way that:
(i) The restriction of the system to its pure simply-typed fragment coincides with the definition of the safe lambda calculus;
(ii) The side-condition of the abstraction and application rules constrains only Γ-variables.
Terms-in-context thus generated are written $\Gamma \vdash_{\mathsf{s}} M : T$.

An immediate consequence is that terms-in-context of the safe fragment satisfy the basic property of the safe lambda calculus:

$$\Gamma \vdash_{\mathsf{s}} M : T \implies \forall z : A \in \Gamma . z \in FV(M) \implies \operatorname{ord} A \geq \operatorname{ord} T \ .$$

Further, in order for this language to be of any use, it must satisfy the subject reduction lemma (*i.e.*, the small-step reduction semantics must preserve safety).
The results of the previous sections show that IA and the recursion-free fragments of PCF both fit in this setting.

3.6 Related work

The safety condition for higher-order grammars

We have mentioned the result of Knapik et al. [KNU02] that infinite trees generated by *safe higher-order recursion schemes* have decidable MSO theories. A natural question is whether the *safety condition* is really necessary. This has been partially answered by Aehlig et al. [AdMO05a] who showed that unrestricted order-2 recursion schemes have decidable MSO theories. Concerning word languages, the same authors have shown [AdMO05b] that level 2 safe higher-order grammars are as expressive as (non-deterministic) unsafe ones. De Miranda's thesis [dM06] proposes a unified framework for the study of higher-order grammars and gives a detailed analysis of the safety constraint at level 2.

More recently, Ong obtained a more general result and showed that the MSO theory of infinite trees generated by higher-order grammars of any level, *whether safe or not*, is decidable [Ong06a]. Using an argument based on innocent game semantics, he establishes a correspondence between the *computation tree* of a higher-order grammar and the *value tree* that it generates: Paths in the value tree correspond to P-views of traversals of the computation tree. Decidability is then obtained by reducing the problem to the acceptance of the (annotated) computation tree by a certain alternating parity tree automaton.

The equivalence of *safe* higher-order grammars and higher-order deterministic pushdown automata for the purpose of generating infinite trees [KNU02] has its counterpart in the general (not necessarily safe) case: Hague et al. [HMOS08] established the equivalence of order-n higher-order grammars and order-n *collapsible pushdown automata*. Those automata form a new kind of pushdown systems in which every stack symbol has a link to a stack situated somewhere below it and with an additional stack operation whose effect is to "collapse" a stack s to the state indicated by the link from the top stack symbol.

Chapter 4

A Concrete Presentation of Game Semantics

This chapter is an independent part of the thesis that concerns game semantics only. One of the key features of game models is that they abstract the syntax: two syntactically different terms that are equivalent have the same denotation. Here we rework the innocent game semantics of Hyland-Ong [HO00] in a more concrete way. We establish an explicit correspondence between the game denotation of a term and its syntax. Our approach follows ideas recently introduced by Ong [Ong06a], namely the notion of *computation tree* of a simply-typed lambda-term and *traversals* over the computation tree. A computation tree is just an abstract syntax tree (AST) representation of the η-long normal form of a term. Traversals are justified sequences of nodes of the computation tree respecting some formation rules. They provide a way to perform *local computation* of β-reductions as opposed to a global approach where β-redexes are contracted using substitution; they can be viewed as an implementation of the *linear head reduction strategy* [DR04].

The culmination of this chapter is the *Correspondence Theorem* (Theorem 4.96) which is in two parts. First it states that traversals over the computation tree are just representations of the uncovering of plays of the strategy-denotation of the term. More precisely there is an isomorphism between the set of traversals and the *revealed* game denotation, where the revealed denotation is computed similarly to the standard strategy denotation except that internal moves are not hidden after composition. To internal moves in plays of game semantics—those hidden when performing strategy composition—correspond "internal nodes" in the traversal setting. The subsequence of a traversal consisting of non-internal nodes only is called the *core* of the traversal. The second part of the theorem states that the standard game denotation of a term is isomorphic to the set traversal cores over its computation tree.

The traversal theory is a top-down approach: the computation tree is traversed from the root and jumps occur from one subterm to another when necessary. This contrasts with the traditional presentation of game semantics where the denotation of a term is calculated bottom-up by composing the denotations of its subterms. Intuitively, the latter approach is less efficient as it requires one to compute the denotation of every subterm even though some of its behaviour will never be exhibited when used as part of the whole term. Another advantage of the traversal theory is that it is flexible enough to accommodate to other functional languages: it is possible to throw in various constants while retaining the game semantic correspondence. At the end of the chapter we show how to adapt the setting to languages like PCF and Idealized Algol.

The definability result of game semantics gives us a strong correspondence between beta-normal terms and their game denotation: given a strategy one can extract from it a term in beta-normal form whose denotation is this particular strategy. The correspondence theorem make this relation between syntax and semantics more precise and generalizes it to terms that are not in beta-normal form. This makes it easier, for instance, to analyze the effect that a syntactic restriction has on the strategy denotation of a term. This is illustrated in the next chapter where we make use of the Correspondence Theorem to analyze the game semantics of the safety restriction. (In Chapter 6 we also give a full analysis of the game semantics of safety without appealing to any syntactic argument.)

Although subsequent results of the thesis rely on the Correspondence Theorem, this chapter is of independent interest. The traversal theory is a new approach to a local analysis of beta reduction [DR93]. It has been applied in a number of different places:

- Ong originally introduced the traversal theory to show decidability of monadic second-order theories of infinite structures generated by higher-order grammars [Ong06a];
- Stirling showed that the higher-order matching problem is decidable. His proof introduced the concept of tree-checking games that are closely related to the theory of traversals [Sti06];
- In a preprint, Ong and Tzevelekos use the traversal theory to study various versions of the Reachability Problem for functionnal programming languages [OT].

Further the inductive definition of traversals lends itself well to automaton characterizations. For instance Hague *et al.* considered the case of higher-order recursion schemes. They show that the set of traversals of the computation tree induced by a recursion scheme can be computed by a *collapsible higher-order pushdown automaton* [HMOS08]. This suggests potential applications in algorithmic game semantics where such characterization permits one to reduce the observational equivalence problem to an automata equivalence problem.

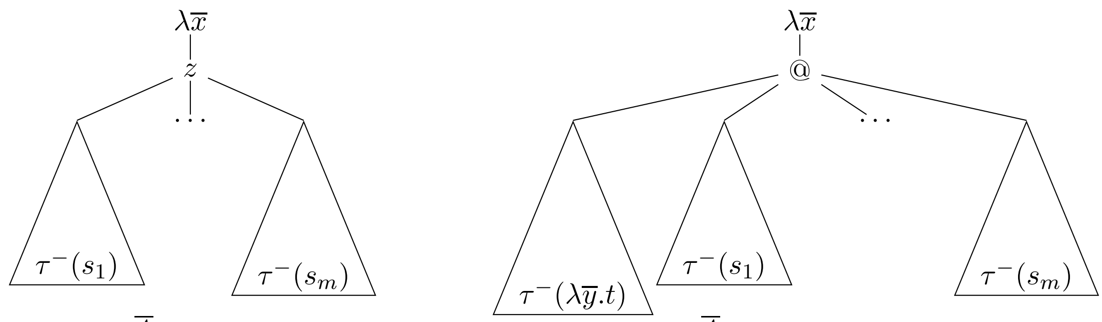
Related works: The useful transference technique between plays and traversals was originally introduced by Ong for studying the decidability of monadic second-order theories of infinite structures generated by higher-order grammars [Ong06a]. In this setting, the Σ-constants or terminal symbols are at most first-order, and are *uninterpreted*. Here we present an extension of this framework to the general case of the simply-typed lambda calculus with free variables of any order. Further the term considered is not required to be of ground type contrary to higher-order grammars. This is achieved by adding new traversal rules to handle term variables whose 'value' is undetermined (*i.e.*, those that cannot be resolved through redex-contraction). We also augment computation trees with additional nodes that account for answer moves of game semantics. This enables our framework to be extended to languages with interpreted constants such as PCF and Idealized Algol.

A notion of local computation of β-reduction has also been investigated through the use of special graphs called "virtual nets" that embed the lambda calculus [DR93].

Asperti et al. introduced [ADLR94] a syntactic representation of lambda-terms based on Lamping's graphs [Lam90]. They unified various notions of paths (regular, legal, consistent and persistent paths) that have appeared in the literature as ways to implement graph-based reduction of lambda-expressions. We can regard a traversal as an alternative notion of path adapted to the graph representation of lambda-expressions given by computation trees.

4.1 Computation tree

We work in the general setting of the simply-typed lambda calculus extended with a fixed set Σ of higher-order constants. For now we assume that the Σ-constants are *uninterpreted*, so we can just think of them as data constructors. (Formally a constant $c \in \Sigma$ is ***uninterpreted*** if the small-step semantics of the language does not contain any rule of the form $c\, M_1 \ldots M_k \rightarrow f_c(M_1, \ldots, M_k)$ for some function f_c over terms $M_1, \ldots, M_k$.) We fix a simply-typed term-in-context $\Gamma \vdash M : T$ for the rest of the section.



$\tau^-(\lambda\overline{x}^A.zs_1\ldots s_m : o)$ $m \geq 0$ and $z \in \mathcal{V} \cup \Sigma$

$\tau^-(\lambda\overline{x}^A.(\lambda y^\tau.t)s_1\ldots s_m : o)$ $m \geq 1$

Table 4.1: The tree $\tau^-(M)$.

4.1.1 Definition

We define the *computation tree* of a simply-typed lambda-term as an abstract syntax tree representation of its *η-long normal form* (Def. 3.29). Our definition generalizes the notion of computation tree for higher-order recursion schemes [Ong06a].

Recall that an η-long normal form is of the form $\lambda\overline{x}.s_0s_1\ldots s_m$ where (i) $\overline{x} = x_1\ldots x_n$ for $n \geq 0$ and $s_0s_1\ldots s_m$ is of ground type, (ii) each s_j for $j \in 1..m$ is in η-long nf; (iii) either s_0 is a variable or a constant and $m \geq 0$, or s_0 is an abstraction $\lambda\overline{y}.s$ and $m \geq 1$ where s is in η-long nf. Note that for terms of ground type the η-long nf is of the form $\lambda.N$. Although the symbol 'λ' does not correspond to a real lambda-abstraction—we call it 'dummy lambda'—it will still be convenient to keep it in expressions representing eta-long normal forms.

Definition 4.1. Let $\Gamma \vdash_{\mathsf{st}} M : T$ be a simply-typed term with variable names from $\mathcal{V}$ and constants from Σ. The *pre-computation* tree $\tau^-(M)$ with labels taken from $\{@\} \cup \Sigma \cup \mathcal{V} \cup \{\lambda x_1\ldots x_n \mid x_1,\ldots,x_n \in \mathcal{V}, n \in \mathbb{N}\}$, is defined inductively on its η-long normal form as follows.

$$\begin{aligned}
\text{For } m \geq 0,\ z \in \mathcal{V}\cup\Sigma\text{: } \tau^-(\lambda\overline{x}^A.zs_1\ldots s_m : o) &= \lambda\overline{x}\langle z\langle\tau^-(s_1),\ldots,\tau^-(s_m)\rangle\rangle \\
\text{for } m \geq 1\text{: } \tau^-(\lambda\overline{x}^A.(\lambda y^\tau.t)s_1\ldots s_m : o) &= \lambda\overline{x}\langle @\langle\tau^-(\lambda y^\tau.t),\tau^-(s_1),\ldots,\tau^-(s_m)\rangle\rangle\ ,
\end{aligned}$$

where we write $l\langle t_1,\ldots,t_n\rangle$, for $n \geq 0$, to denote the *ordered tree* whose root is labelled l and has n child-subtrees t_1, ..., t_n. The trees from the equations above are illustrated in Table 4.1.

By convention the first level of a tree (where the root lies) is numbered 0. In the tree $\tau^-(M)$, odd-levels contain variable, constant and application nodes; even-levels contain only λ-nodes. A single λ-node can represent several consecutive abstractions or it can just be a *dummy lambda* (if the corresponding subterm is of ground type).

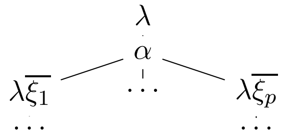
Definition 4.2. Let M be a simply-typed term not necessarily in η-long normal form. Let $\mathcal{D}$ denote the set of values of the base type o. The ***computation tree*** of M, written $\tau(M)$ is the tree obtained from $\tau^-(M)$ by attaching leaves to each node as follows: for every node $n \in \tau^-(M)$, the corresponding node in $\tau(M)$ has a child leaf labelled v_n, called ***value-leaf***, for every possible value $v \in \mathcal{D}$.

Inner nodes of the tree are thus of three kinds:

- λ-nodes labelled $\lambda\overline{x}$ for some list of variables $\overline{x}$ (Note that a λ-node represents several consecutive variable abstractions),
- application nodes labelled @,
- variable or constant nodes with labels in $\Sigma \cup \mathcal{V}$.

The 0^{th} child of an @-node is called a ***prime node***. The children inner nodes of @-nodes and Σ-nodes are called ***spawn nodes***.

Example 4.3.

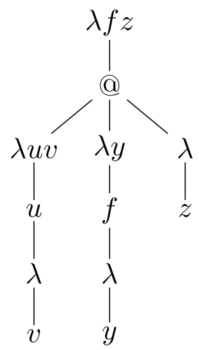
- The computation tree of a ground type variable or constant α is  ;
- The computation tree of a higher-order variable or constant $\alpha : (A_1, \dots, A_p, o)$ has the following form:  ;

Example 4.4. Take $\vdash_{\mathsf{st}} \lambda f^{o\to o}.(\lambda u^{o\to o}.u)f : (o \to o) \to o \to o$.

Its η-long normal form is:

$$\begin{aligned}\vdash_{\mathsf{st}} & \lambda f^{o\to o} z^o. \\ & (\lambda u^{o\to o} v^o.u(\lambda.v)) \\ & (\lambda y^o.fy) \\ & (\lambda.z) \\ : & (o \to o) \to o \to o\end{aligned}$$

Its computation tree is:

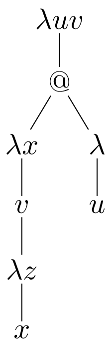


Example 4.5. Take $\vdash_{\mathsf{st}} \lambda u^o v^{((o\to o)\to o)}.(\lambda x^o.v(\lambda z^o.x))u : o \to ((o \to o) \to o) \to o$.

Its η-long normal form is:

$$\begin{aligned}\vdash_{\mathsf{st}} & \lambda u^o v^{((o\to o)\to o)}. \\ & (\lambda x^o.v(\lambda z^o.x))u \\ : & o \to ((o \to o) \to o) \to o\end{aligned}$$

Its computation tree is:



Notations 4.6 We write $\circledast$ to denote the root of $\tau(M)$. The set of nodes of the tree is denoted by N. The subset of *inner nodes* is denoted IN and the subset of *leaf nodes* is denoted L (Thus $N = IN \cup L$). We write $E \subseteq N \times N$ to denote the parent-child relation on the tree nodes.

We write IN_Σ for the set of Σ-labelled (inner) nodes, $IN_@$ for the set of @-labelled nodes, IN_{var} for the set of variable nodes, IN_{fv} for the subset of IN_{var} consisting of free-variable nodes, IN_{prime} for the set of prime nodes and IN_{spawn} for the set of spawn nodes.

For \$ ranging over $\{@, \lambda, \mathsf{var}, \mathsf{fv}\}$, we write $L_\$$ to denote the set of value-leaves which are children of nodes from $IN_\$$; formally $L_\$ = \{v_n \mid n \in IN_\$, v \in \mathcal{D}\}$. We write $N_\$$ for $IN_\$ \cup L_\$$.

Further we partition the set of nodes in two subsets: the ***P-nodes*** $IN_{\mathsf{var}} \cup IN_\Sigma \cup IN_@ \cup L_\lambda$, and the ***O-nodes*** $L_{\mathsf{var}} \cup L_\Sigma \cup L_@ \cup IN_\lambda$.

Each subtree of the computation tree $\tau(M)$ represents a subterm of $\lceil M \rceil$. For every lambda node n in IN_λ we write $M^{(n)}$ for the subterm of $\lceil M \rceil$ corresponding to the subtree of $\tau(M)$

rooted at n, and $N^{(n)}$ for the set of nodes of this subtree (which is isomorphic to $\tau(M^{(n)})$); formally $N^{(n)} = E^*(\{n\})$ where E^* denotes the transitive, reflexive closure of the parent-child relation E. (In particular we have $M^{(\circledast)} = \lceil M \rceil$.)

Remark 4.7 Since the computation tree is computed from the eta-long normal form, for every subtree of $\tau(M)$ of the form

$$\begin{array}{ccccc} & & \lambda\overline{\varphi} & & \\ & & n & & \\ \lambda\overline{\xi_1} & & \ldots & & \lambda\overline{\xi_p} \\ \ldots & & & & \ldots \end{array}$$

, we have $\operatorname{ord} M^{(n)} = 0$.

Definition 4.8 (Type and order of a node)**.** Suppose $\Gamma \vdash M : T$. The ***type*** of an inner-node $n \in IN$ of $\tau(M)$ written $\mathsf{type}(n)$ is defined as follows:

$$\begin{array}{rcl} \mathsf{type}(\circledast) & = & \Gamma \rightarrow T, \\ \text{for } n \in (IN_\lambda \cup IN_@) \setminus \{\circledast\}\text{: } \mathsf{type}(n) & = & \text{type of the term } M^{(n)}, \\ \text{for } n \in IN_{\mathsf{var}} \cup IN_\Sigma\text{: } \mathsf{type}(n) & = & \text{type of the variable labelling } n. \end{array}$$

where the notation $\Gamma \rightarrow T$ is an abbreviation for $(A_1, \ldots, A_p, T)$ and $A_1, \ldots, A_p$ are the types of the variables in the context Γ.

The ***order*** of a node n, written $\operatorname{ord} n$, is defined as follows: a value-leaf $v \in L$ has order 0 and the order of an inner node $n \in IN$ is defined as the order of its type. In particular, the type of a lambda node different from the root is the type of the term represented by the sub-tree rooted at that node, and the type of a variable-node is the type of the variable labelling it.

Since the computation tree is calculated from the η-long normal form, all the @-nodes have order 0 ($\operatorname{ord} @ = 0$); for every lambda node $\lambda\overline{\xi} \neq \circledast$ we have $\operatorname{ord} \lambda\overline{\xi} = 1 + \max_{z \in \overline{\xi}} \operatorname{ord} z$; and if the root $\circledast$ is labelled $\lambda\overline{\xi}$ then $\operatorname{ord} \circledast = 1 + \max_{z \in \overline{\xi} \cup \Gamma} \operatorname{ord} z$ with the convention $\max \emptyset = -1$.

Definition 4.9 (Binder)**.** We say that a variable node n labelled x is ***bound*** by a node m, and m is called the ***binder*** of n, if m is the closest node in the path from n to the root such that m is labelled $\lambda\overline{\xi}$ with $x \in \overline{\xi}$.

4.1.2 Pointers and justified sequence of nodes

4.1.2.1 Definitions

Definition 4.10 (Enabling)**.** The ***enabling relation*** $\vdash$ is defined on the set of nodes of the computation tree as follows. We write $m \vdash n$ and we say that m enables n if and only if $m \in L \cup IN_\lambda \cup IN_{\mathsf{var}}$ and one of the following conditions holds:

- $n \in IN_{\mathsf{fv}}$ and m is the root $\circledast$;
- $n \in IN_{\mathsf{var}} \setminus IN_{\mathsf{fv}}$ and m is n's binder, in which case we write $m \vdash_i n$ to indicate that n is the i^{th} variable bound by m;
- $n \in IN_\lambda$ and m is n's parent;
- $n \in L$ and m is n's parent (*i.e.*, $n = v_m$ for some $v \in \mathcal{D}$).

Formally:

$$\begin{array}{rcl} \vdash & = & \{(\circledast, n) \mid n \in IN_{\mathsf{fv}}\} \\ & & \cup\ \{(\lambda\overline{x}, x) \mid x \in IN_{\mathsf{var}} \setminus IN_{\mathsf{fv}} \wedge \lambda\overline{x} \text{ is } x\text{'s binder}\} \\ & & \cup\ \{(m, \lambda\overline{\eta}) \mid m \text{ is } \lambda\overline{\eta}\text{'s parent and } \lambda\overline{\eta} \in IN_\lambda\} \\ & & \cup\ \{(m, v_m) \mid v \in \mathcal{D}, m \in IN\} \ . \end{array}$$

Note that in particular, free variable nodes are enabled by the root. Table 4.2 recapitulates the possible node types for the enabler node depending on the type of n.

If $n \in \_$	then $m \in \_$
IN_λ	$IN_{\mathsf{var}} \cup IN_\Sigma \cup IN_@$
L_{var}	IN_{var}
$L_@$	$IN_@$
L_Σ	IN_Σ
IN_{var}	IN_λ
IN_Σ	n.a.
$IN_@$	n.a.
L_λ	IN_λ

Table 4.2: Type of the enabler node in "$m \vdash n$".

Nodes that are not in the image of the relation $\vdash$ are called ***initial nodes***; the initial P-nodes are $IN_@ \cup IN_\Sigma$ and the only initial O-node is the root $\circledast$.

We say that a node n_0 of the computation tree is ***hereditarily enabled*** by $n_p \in N$ if there are nodes $n_1, \ldots, n_{p-1} \in N$ such that n_{i+1} enables n_i for all $i \in 0..p-1$.

For every sets of nodes $S, H \subseteq N$ we write $S^{H\vdash}$ to denote the subset $S \cap \vdash^* (H)$ of S consisting of nodes hereditarily enabled by some node in H. Formally:

$$S^{H\vdash} = \{n \in S \mid \exists m \in H \text{ s.t. } m \vdash^* n\} \ .$$

If H is a singleton $\{m\}$ then we abbreviate $S^{\{m\}\vdash}$ into $S^{m\vdash}$.

It can be verified that a non-initial node is either hereditarily enabled by the root, an application node or a constant node; thus the subsets $\{\circledast\}$, $IN_@$, $IN_@$, $N^{\circledast\vdash}$, $N^{IN_@\vdash}$ and $N^{IN_\Sigma\vdash}$ form a partition of N. The elements of $IN_{\mathsf{var}}^{\circledast\vdash}$ (*i.e.*, variable nodes that are hereditarily enabled by the root of $\tau(M)$) are called ***input-variables nodes***.

We use the following numbering conventions: The first child of a @-node—a prime node—is numbered 0; the first child of a variable or constant node is numbered 1; and variables in $\overline{\xi}$ are numbered from 1 onward ($\overline{\xi} = \xi_1 \ldots \xi_n$). We write $n.i$ to denote the i^{th} child of node n.

Definition 4.11 (Justified sequence of nodes)**.** A ***justified sequence of nodes*** is a sequence of nodes s of the computation tree $\tau(M)$ with pointers. Each occurrence in s of a node n in $L \cup IN_\lambda \cup IN_{\mathsf{var}}$ has a link pointing to some preceding occurrence of a node m satisfying $m \vdash n$; and occurrences of nodes in $IN_@ \cup IN_\Sigma$ do not have pointer.

If an occurrence n points to an occurrence m in s then we say that m ***justifies*** n. If n is an inner node then we represent this pointer in the sequence as $\overset{i}{\overparen{m \ldots n}}$ where the label indicates that either n is labelled with the i^{th} variable abstracted by the λ-node m or that n is the i^{th} child of m. The pointer associated to a leaf v_m, for some value $v \in \mathcal{D}$ and internal node $m \in IN$, is represented as $\overset{v}{\overparen{m \cdot \ldots \cdot v_m}}$.

To sum-up, a pointer in a justified sequence of nodes has one of the following forms:

	$\overparen{r \cdot \ldots \cdot z}$	for some occurrence r of $\tau(M)$'s root and $z \in IN_{\mathsf{fv}}$;
or	$\overset{i}{\overparen{\lambda\overline{\xi} \cdot \ldots \cdot \xi_i}}$	for some variable ξ_i bound by $\lambda\overline{\xi}$, $i \in 1..\lvert\overline{\xi}\rvert$;
or	$\overset{j}{\overparen{@ \cdot \ldots \cdot \lambda\overline{\eta}}}$	$j \in \{1..(arity(@) - 1)\}$;
or	$\overset{k}{\overparen{\alpha \cdot \ldots \cdot \lambda\overline{\eta}}}$,	for $\alpha \in IN_\Sigma \cup IN_{\mathsf{var}}$, $k \in \{1..arity(\alpha)\}$;
or	$\overset{v}{\overparen{m \cdot \ldots \cdot v_m}}$	for some value $v \in \mathcal{D}$ and inner node $m \in IN$.

We say that an inner node n in of a justified sequence of nodes is ***answered***[1] by the value-leaf v_n if there is an occurrence of v_n for some value v in the sequence that points to n, otherwise we say that n is ***unanswered***. The last unanswered node is called the ***pending node***. A justified sequence of nodes is ***well-bracketed*** if each value-leaf occurring in it is justified by the pending node at that point.

For every justified sequence of nodes t we write $?(t)$ to denote the subsequence of t consisting only of unanswered nodes. Formally:

$$\begin{aligned} ?(u_1 \cdot n \cdot u_2 \cdot v_n) &= ?(u_1 \cdot n \cdot u_2) \setminus \{n\} && \text{for some value } v \in \mathcal{D} \quad , \\ ?(u \cdot n) &= ?(u) \cdot n && \text{for } n \notin L \quad , \end{aligned}$$

where $u \setminus \{n\}$ denotes the subsequence of u obtained by removing the occurrence n.

If u is a well-bracketed sequences then $?(u)$ can be defined as follows:

$$\begin{aligned} ?(u \cdot n \ldots v_n) &= ?(u) && \text{for some value } v \in \mathcal{D} \ , \\ ?(u \cdot n) &= ?(u) \cdot n && \text{where } n \notin L \ . \end{aligned}$$

Notations 4.12 We write $s = t$ to denote that the justified sequences s and t have the same nodes *and* pointers. Justified sequence of nodes can be ordered using the prefix ordering: $t \leqslant t'$ if and only if $t = t'$ or the sequence of nodes t is a finite prefix of t' (and the pointers of t are the same as the pointers of the corresponding prefix of t'). Note that with this definition, infinite justified sequences can also be compared. This ordering gives rise to a complete partial order. We say that a node n_0 of a justified sequence is ***hereditarily justified*** by n_p if there are nodes $n_1, n_2, \ldots n_{p-1}$ in the sequence such that n_i points to n_{i+1} for all $i \in \{0..p-1\}$. We write t^ω to denote the last element of the sequence t.

4.1.2.2 Projection

We define two different projection operations on justified sequences of nodes.

Definition 4.13 (Projection on a set of nodes)**.** Let A be a subset of N, the set of nodes of $\tau(M)$, and t be a justified sequence of nodes then we write $t \upharpoonright A$ for the subsequence of t consisting of nodes in A. This operation can cause a node n to lose its pointer. In that case we reassign the target of the pointer to the last node in $t_{\leqslant n} \upharpoonright A$ that hereditarily justifies n (This node can be found by following the pointers from n until reaching a node appearing in A); if there is no such node then n just loses its pointer.

Definition 4.14 (Hereditary projection)**.** Let t be a justified sequence of nodes of $\tau(M)$ and n be some occurrence in t. We define the justified sequence $t \upharpoonright n$ as the subsequence of t consisting of nodes hereditarily justified by n in t.

Lemma 4.15. *The projection function* $\_ \upharpoonright n$ *defined on the cpo of justified sequences ordered by the prefix ordering is continuous.*

Proof. Clearly $\_ \upharpoonright n$ is monotonous. Suppose that $(t_i)_{i \in \omega}$ is a chain of justified sequences. Let u be a finite prefix of $(\bigvee t_i) \upharpoonright n$. Then $u = s \upharpoonright n$ for some finite prefix s of $\bigvee t_i$. Since s is finite we must have $s \leqslant t_j$ for some $j \in \omega$. Therefore $u \leqslant t_j \upharpoonright n \leqslant \bigvee (t_j \upharpoonright n)$. This is valid for every finite prefix u of $(\bigvee t_i) \upharpoonright n$ thus $(\bigvee t_i) \upharpoonright n \leqslant \bigvee (t_j \upharpoonright n)$. $\square$

The nodes occurrences that do not have pointers in a justified sequence are called ***initial occurrences***. An initial occurrence is either the root of the computation tree, an @-node or a Σ-node. Let n be occurrence in a justified sequence of nodes t. The subsequence of t consisting of occurrences that are hereditarily justified by the same *initial occurrence* as n is called ***thread*** of n. Thus each thread in a justified sequence contains a single initial occurrence. The thread of n is given by $n \upharpoonright i$ where i is the first node in t hereditarily justifying n; i is called the ***initial occurrence of the thread of*** n.

[1] This terminology is deliberately suggestive of the correspondence with game-semantics.

4.1.2.3 Views

The notion of ***P-view*** $\ulcorner t \urcorner$ of a justified sequence of nodes t is defined the same way as the P-view of a justified sequences of moves in game semantics:

Definition 4.16 (P-view of justified sequence of nodes)**.** The P-view of a justified sequence of nodes t of $\tau(M)$, written $\ulcorner t \urcorner$, is defined as follows:

$$
\begin{array}{rcll}
\ulcorner \epsilon \urcorner & = & \epsilon & \\
\ulcorner s \cdot n \urcorner & = & \ulcorner s \urcorner \cdot n & \text{if } n \text{ is a P-node } ; \\
\ulcorner s \cdot m \cdot \ldots \cdot n \urcorner & = & \ulcorner s \urcorner \cdot m \cdot n & \text{if } n \text{ is an O-node } ; \\
\ulcorner s \cdot \circledast \urcorner & = & \circledast \ . &
\end{array}
$$

The equalities in the definition determine pointers implicitly. For instance in the second clause, if in the left-hand side n points to some node in s that is also present in $\ulcorner s \urcorner$ then in the right-hand side n points to that occurrence of the node in $\ulcorner s \urcorner$.

The O-view of s, written $\llcorner s \lrcorner$, is defined dually.

Definition 4.17 (O-view of justified sequence of nodes)**.** The O-view of a justified sequence of nodes t of $\tau(M)$, written $\llcorner t \lrcorner$, is defined as follows:

$$
\begin{array}{rcll}
\llcorner \epsilon \lrcorner & = & \epsilon & \\
\llcorner s \cdot n \lrcorner & = & \llcorner s \lrcorner \cdot n & \text{if } n \text{ is an O-node } ; \\
\llcorner s \cdot m \cdot \ldots \cdot n \lrcorner & = & \llcorner s \lrcorner \cdot m \cdot n & \text{if } n \text{ is a non-initial P-node } ; \\
\llcorner s \cdot n \lrcorner & = & n & \text{if } n \text{ is an initial P-node } .
\end{array}
$$

We borrow some terminology from game semantics:

Definition 4.18. Let s be a justified sequence of nodes. We list the following axioms:

- ***Alternation***: for every pair of consecutive nodes in s, one is a P-node and the other is an O-node;
- ***P-visibility***: for every occurrence in s of a non-initial P-node, its justifier occurs in the P-view at that point;
- ***O-visibility***: for every occurrence in s of a non-initial O-node, its justifier occurs in the O-view at that point.

We then have the same basic property as in game semantics: The P-view (resp. O-view) of a justified sequence satisfying P-visibility (resp. O-visibility) is a well-formed justified sequence satisfying P-visibility (resp. P-visibility). (This property follows by an easy induction.)

4.1.3 Traversal of the computation tree

We now define the notion of *traversal* over the computation tree $\tau(M)$. We first consider the simply-typed lambda calculus without interpreted constants; everything remains valid in the presence of *uninterpreted* constants as we can just consider them as free variables. In the next section, we extend the notion of traversal to a more general setting with interpreted constants.

4.1.3.1 Traversals for simply-typed λ-terms

Informally, a traversal is a justified sequence of nodes of the computation tree where each node indicates a step that is taken during the evaluation of the term.

Definition 4.19 (Traversals for simply-typed lambda-terms)**.** The set $\mathcal{T}rav(M)$ of ***traversals*** over $\tau(M)$ is defined by induction over the rules of Table 4.3. A traversal that cannot be extended by any rule is said to be *maximal*.

Initialization rules

(Empty) $\epsilon \in \mathcal{T}rav(M)$.

(Root) The sequence consisting of a single occurrence of $\tau(M)$'s root is a traversal.

Structural rules

(Lam) If $t \cdot \lambda\overline{\xi}$ is a traversal then so is $t \cdot \lambda\overline{\xi} \cdot n$ where n denotes $\lambda\overline{\xi}$'s child in $\tau(M)$ and:

- If $n \in IN_{@} \cup IN_{\Sigma}$ then it has no justifier;
- if $n \in IN_{\mathsf{var}} \setminus IN_{\mathsf{fv}}$ then it points to the only occurrence[a] of its binder in $\ulcorner t \cdot \lambda\overline{\xi} \urcorner$;
- if $n \in IN_{\mathsf{fv}}$ then it points to the only occurrence of the root $\circledast$ in $\ulcorner t \cdot \lambda\overline{\xi} \urcorner$.

(App) If $t \cdot @$ is a traversal then so is $t \cdot @ \cdot n$.

Input-variable rules

(InputVar) If t is a traversal where $t^{\omega} \in IN_{\mathsf{var}}^{\circledast\vdash} \cup L_{\lambda}^{\circledast\vdash}$ and x is an occurrence of a variable node in $\llcorner t \lrcorner$ then so is $t \cdot n$ for every child λ-node n of x, n pointing to x.

(InputValue) If $t_1 \cdot x \cdot t_2$ is a traversal with pending node $x \in IN_{\mathsf{var}}^{\circledast\vdash}$ then so is $t_1 \cdot x \cdot t_2 \cdot v_x$ for all $v \in \mathcal{D}$.

Copy-cat rules

(Var) If $t \cdot n \cdot \lambda\overline{x} \dots x_i$ is a traversal where $x_i \in IN_{\mathsf{var}}^{@\vdash}$ then so is $t \cdot n \cdot \lambda\overline{x} \dots x_i \cdot \lambda\overline{\eta_i}$.

(Value) If $t \cdot m \cdot n \dots v_n$ is a traversal where $n \in IN$ then so is $t \cdot m \cdot n \dots v_n \cdot v_m$.

Table 4.3: Traversal rules for the simply-typed lambda calculus.

[a] Prop. 4.29 will show that P-views are paths in the tree thus n's enabler occurs exactly once in the P-view.

Example 4.20. The following justified sequence is a traversal of the computation tree from Example 4.4:

$$t = \lambda fz \cdot @ \cdot \lambda uv \cdot u \cdot \lambda y \cdot f \cdot \lambda \cdot y \cdot \lambda \cdot v \cdot \lambda \cdot z \ .$$

Remark 4.21

1. The rule (Value) from Table 4.3 can be equivalently reformulated into four distinct rules ($\mathsf{Value}^{\lambda\mapsto @}$), ($\mathsf{Value}^{@\mapsto\lambda}$), ($\mathsf{Value}^{\lambda\mapsto\mathsf{var}}$) and ($\mathsf{Value}^{\mathsf{var}\mapsto\lambda}$), each one dealing with a different possible category for the nodes n and m:

   ($\mathsf{Value}^{\lambda\mapsto @}$) If $t \cdot @ \cdot \lambda\overline{z} \ldots v_{\lambda\overline{z}}$ is a traversal then so is $t \cdot @ \cdot \lambda\overline{z} \ldots v_{\lambda\overline{z}} \cdot v_{@}$.

   ($\mathsf{Value}^{@\mapsto\lambda}$) If $t \cdot \lambda\overline{\xi} \cdot @ \ldots v_{@}$ is a traversal then so is $t \cdot \lambda\overline{\xi} \cdot @ \ldots v_{@} \cdot v_{\lambda\overline{\xi}}$.

   ($\mathsf{Value}^{\lambda\mapsto\mathsf{var}}$) If $t \cdot y \cdot \lambda\overline{\xi} \ldots v_{\lambda\overline{\xi}}$ is a traversal with $y \in IN^{@\vdash}_{\mathsf{var}}$ then so is $t \cdot y \cdot \lambda\overline{\xi} \ldots v_{\lambda\overline{\xi}} \cdot v_y$.

   ($\mathsf{Value}^{\mathsf{var}\mapsto\lambda}$) If $t \cdot \lambda\overline{\xi} \cdot x \ldots v_x$ is a traversal where $x \in IN_{\mathsf{var}}$ then so is $t \cdot \lambda\overline{\xi} \cdot x \ldots v_x \cdot v_{\lambda\overline{\xi}}$.

   In the rest of this chapter we will prove various resulting by induction on the structure of a traversal and by case analysis on the last rule used to form it. Some of these proofs will rely on the above reformulation rather than the original definition of (Value).

2. In the rule ($\mathsf{InputValue}$), the last node in the traversal $t_1 \cdot x \cdot t_2$ necessarily belongs to $IN_{\mathsf{var}} \cup L_\lambda$. Indeed, since the pending node x is a variable node, the traversal is of the form

   $$\ldots \cdot x \cdot \lambda\overline{\eta}_1 \ldots v^1_{\lambda\overline{\eta}_1} \lambda\overline{\eta}_2 \ldots v^2_{\lambda\overline{\eta}_2} \ldots \lambda\overline{\eta}_k \ldots v^k_{\lambda\overline{\eta}_k}$$

   for some nodes $\lambda\overline{\eta}_k$, values $v^k \in \mathcal{D}$ and $k \geq 0$; thus the last occurrence belongs to IN_{var} if $k = 0$ and to L_λ if $k \geq 1$.

   Furthermore, the pending node appears necessarily in the O-view.

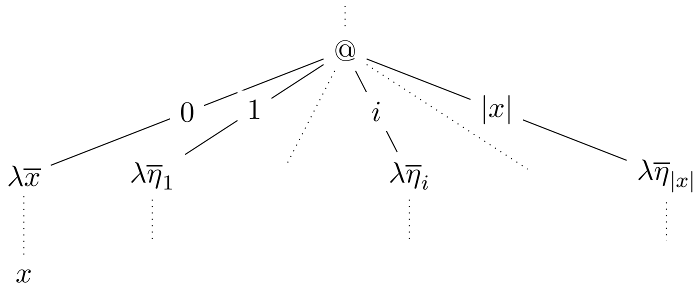
   These two observations show that the rule ($\mathsf{InputValue}$) is essentially a specialization of ($\mathsf{InputVar}$) to value-leaves. The only difference is that ($\mathsf{InputVar}$) allows the visited node to be justified by *any* variable node occurring in the O-view whereas ($\mathsf{InputValue}$) constrains the node to be justified by the pending node (which necessarily occurs in the O-view). This additional restriction is required to ensure that traversals are well-bracketed.

3. In the rule (Value), it is possible to replace "$n \in IN$" by the stronger condition "$n \in IN \setminus IN^{\circledast\vdash}_{\lambda}$". Indeed a later result (Lemma 4.32) will show that if n belongs to $IN^{\circledast\vdash}_{\lambda}$ then the preceding occurrence m is necessarily an input-variable. Furthermore, another result (Prop. 4.29) shows that traversals are well-bracketed, therefore m is necessarily the pending node. Thus in this particular case we can use the rule ($\mathsf{InputValue}$) in place of (Value) to visit v_m.

   The advantage of this alternative formulation is that the traversal rules have disjoint domains of definition.

A traversal always starts with the root node and mainly follows the structure of the tree. The exception is the (Var) rule which permits the traversal to jump across the computation tree. The idea is that after visiting a non-input variable node x, a traversal can jump to the node corresponding to the subterm that would be substituted for x if all the β-redexes occurring in the term were to be reduced. Let $\lambda\overline{x}$ be x's binder and suppose x is the i^{th} variable in $\overline{x}$. The binding node necessarily occurs previously in the traversal (This will be proved in Prop. 4.29). Since x is not hereditarily justified by the root, $\lambda\overline{x}$ is not the root of the tree and therefore it is not the first node of the traversal. We do a case analysis on the node preceding $\lambda\overline{x}$:

- If it is an @-node then $\lambda\overline{x}$ is necessarily the first child node of that node and it has exactly $|\overline{x}|$ siblings:



  In that case, the next step of the traversal is a jump to $\lambda\overline{\eta_i}$—the i^{th} child of @—which corresponds to the subterm that would be substituted for x if the β-reduction was performed:

$$t' \cdot @ \cdot \lambda\overline{x} \cdot \ldots \cdot x \cdot \lambda\overline{\eta_i} \cdot \ldots \in \mathcal{T}rav(M) \ .$$

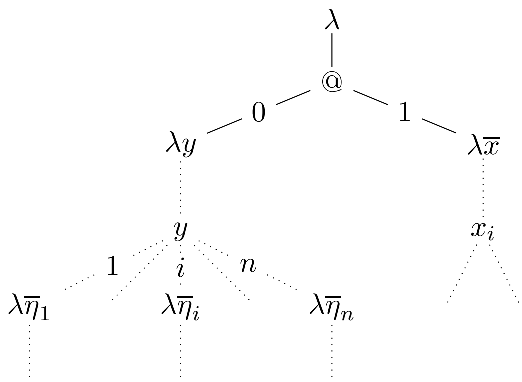
- If it is a variable node y, then the node $\lambda\overline{x}$ was necessarily added to the traversal $t_{\leq y}$ using the (Var) rule. (Indeed, if it was visited using (InputVar) then $\lambda\overline{x}$ would be hereditarily justified by the root, but this is not possible since x_i, bound by $\lambda\overline{x}$, is not an input-variable.) Therefore y is substituted by the term rooted at $\lambda\overline{x}$ during the evaluation of the term. Consequently, during reduction, the variable x will be substituted by the subterm represented by the i^{th} child node of y. We therefore have the traversal

$$t' \cdot y \cdot \lambda\overline{x} \cdot \ldots \cdot x \cdot \lambda\overline{\eta_i} \cdot \ldots$$

Remark 4.22 Our notions of computation tree and traversal differ slightly from the original definitions by Ong [Ong06a]. In his setting:

- computation trees contain (uninterpreted first-order) constants. Here we do not consider constants but, as previously observed, they are automatically accounted for since uninterpreted constants can just be regarded as free variables.
- constants are restricted to order one at most. (Terms are used as generators of trees where first-order constants act as tree-node constructors). Here we do not need this restriction: as long as constants are uninterpreted we can regard them as free variables, even at higher-orders.
- a single rule (Sig) suffices to model the first-order constants. In contrast our setting accounts for higher-order variables, thus the more complicated rules (InputValue) and (InputVar) are required.
- computation trees do not have value-leaves. These are not necessary to model the pure simply-typed lambda calculus. There will be necessary, however, when it comes to model *interpreted* constants such as those of PCF or IA.

Example 4.23. Consider the following computation tree:



An example of traversal of this tree is:

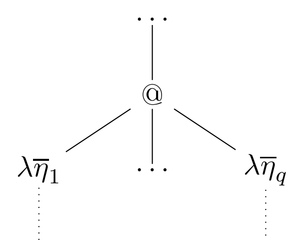
$$\lambda \cdot @ \cdot \lambda y \cdot \ldots \cdot y \cdot \lambda \overline{x} \cdot \ldots \cdot x_i \cdot \lambda \overline{\eta_i} \cdot \ldots$$

Lemma 4.24. *Take a traversal t ending with an inner node hereditarily justified by an application node @. Then the thread of t^ω has the following shape:*

$$@ \cdot \lambda\overline{\xi}_0 \ldots x_1 \cdot \lambda\overline{\xi}_1 \ldots x_2 \cdot \lambda\overline{\xi}_2 \ldots x_3 \cdot \lambda\overline{\xi}_3 \ldots x_4 \ldots x_{k-1}\lambda\overline{\xi}_{k-1} \ldots x_k\lambda\overline{\xi}_k \ .$$

where all the represented nodes appear in the O-view.

Suppose that the initial node @ occurs in the computation as follows:



Let τ_i denote the sub-tree rooted at $\lambda\overline{\eta}_i$ for $i \in \{1..q\}$. Then for every $j \in \{1..k\}$, x_j and $\lambda\overline{\xi}_j$ must belong to two different subtrees τ_i and $\tau_{i'}$. Furthermore, x_j is hereditarily justified by some occurrence of $\lambda\overline{\eta}_i$ in t and $\lambda\overline{\xi}_j$ is hereditarily justified by some occurrence of $\lambda\overline{\eta}_{i'}$ in t (and therefore $\lambda\overline{\xi}_j \in N^{\lambda\overline{\eta}_i\vdash}$ and $x_j \in N^{\lambda\overline{\eta}_{i'}\vdash}$).

Proof. The proof is by an easy induction. □

4.1.3.2 Traversal rules for interpreted constants

The framework that we have established up to now aims at providing a computation model of simply-typed lambda-terms. It is possible to extend it to other extensions of the simply-typed lambda calculus. This is done by completing the traversal rules from Table 4.3 with new rules describing the behaviour of the interpreted constants of the language considered. For instance extending the framework to PCF requires one to define rules for the interpreted constant `cond` that replicate the behaviour of the conditional operation. (In a forthcoming section of this chapter we will give a complete definition of the constant traversal rules for PCF and IA.)

We mentioned before that uninterpreted constants can be regarded as free variables. In the same way, we can consider interpreted constants as a *generalization* of free variables: for both of them, the "code" describing their computational behaviour is not defined within the scope of the term, it is instead assumed that the environment knows how to interpret them. Free variables, however, are more restricted than interpreted constants: When evaluating an applicative term with a free variable in head position, the evaluation of the head variable does not depend on the result of the evaluation of its parameters; whereas for applicative term with an interpreted constant in head position, the outcome of the evaluation may depend on the result of the evaluation of its parameters (*e.g.*, the PCF constant `cond` branches between two control points depending on the result of the evaluation of its first parameter).

We can thus derive a prototype for constant traversal rules by generalizing the input-variable rules (InputValue) and (InputVar):

Definition 4.25 (Constant traversal rule)**.** A ***constant traversal*** has one of the following two forms:

$$(\Sigma\text{-Value})\ \frac{t = t_1 \cdot \alpha \cdot t_2 \in \mathcal{T}rav(M) \quad \alpha \in IN_\Sigma \cup IN_{\mathsf{var}}^{IN_\Sigma\vdash} \quad ?(t)^\omega = \alpha \quad P(t)}{t' = t_1 \cdot \alpha \cdot t_2 \cdot v(t) \in \mathcal{T}rav(M)}$$

or

$$(\Sigma)/(\Sigma\text{-Var})\ \frac{t \in \mathcal{T}rav(M) \quad t^\omega \in IN_\Sigma \cup IN^{IN_\Sigma\vdash} \cup L_\lambda \quad P(t)}{t \cdot n(t) \in \mathcal{T}rav(M)}$$

where:

- $P(t)$ is a predicate expressing some condition on t;
- $v(t)$ is a value-leaf of the node α that is determined by the traversal t;
- $n(t)$ is a lambda node determined by t, and its link—also determined by t—points to some occurrence of its parent node in $\llcorner t \lrcorner$.

Clearly, such rules preserve well-bracketing, alternation and visibility.

Remark 4.26 The extra power of the constant rules over the input-variable rules (InputValue) and (InputVar) comes from their ability to base their choice of next visited node on the shape of the traversal t.

From now on, to make our argument as general as possible, we consider a simply-typed lambda calculus language extended with higher-order interpreted constants for which some constant traversal rules have been defined (in the sense of Def. 4.25). Furthermore, we complete the set of rules with the following additional copy-cat rule:

$$(\mathsf{Value}^{\Sigma\mapsto\lambda})\ t \cdot \lambda\overline{\xi} \cdot c \ldots v_c \in \mathcal{T}rav(M) \wedge\ c \in \Sigma \quad\Longrightarrow\quad t \cdot \lambda\overline{\xi} \cdot c \ldots v_c \cdot v_{\lambda\overline{\xi}} \in \mathcal{T}rav(M)\ .$$

Definition 4.27. A constant traversal rules is ***well-behaved*** if for every traversal $t \cdot \alpha \cdot u \cdot n$ formed with the rule we have $?(u) = \epsilon$.

An example is the rule (Σ-Value) which is well-behaved due to the fact that traversals are well-bracketed. The rule $(\Sigma)/(\Sigma\text{-Var})$, however, is not well-behaved since the node $n(t)$ does not necessarily points to the pending node in t.

Lemma 4.28. *If Σ-constants have order 1 at most, then constant rules are necessarily all well-behaved.*

Proof. In the computation tree, an order-1 constant hereditarily enables only its immediate children (which are all dummy lambda nodes λ). Hence a traversal formed with the rule $(\Sigma)/(\Sigma\text{-Var})$ is of the form:

$$t = \ldots \cdot \alpha \cdot u \cdot \lambda$$

where α appears in $\llcorner t \lrcorner$.

If $u = \epsilon$ then the result trivially holds. Otherwise, u's first node has necessarily been visited with the rule $(\Sigma)/(\Sigma\text{-Var})$ thus u's first node is a dummy lambda node λ' pointing to α. Since α occurs in $\llcorner t \lrcorner$ and since the node λ' enables only its value-leaf in the computation tree, t must be of the following shape:

$$t = \ldots \cdot \alpha \cdot \underbrace{\lambda' \ldots v_{\lambda'} \ldots \lambda}_{u}$$

for some value-leaf $v_{\lambda'}$ of λ'. Again, the node following $v_{\lambda'}$ must be a dummy lambda node pointing to α. By iterating the same argument we obtain that the segment u is a repetition of segments of the form $\lambda' \cdot \ldots v_{\lambda'}$. Hence $?(u) = \epsilon$. ☐

4.1.3.3 Property of traversals

Proposition 4.29. *Let t be a traversal. Then:*

(i) t is a well-defined justified sequence satisfying alternation, well-bracketing, P-visibility and O-visibility;

(ii) If the last element of t is not a value-leaf whose parent-node is a lambda node (i.e., $t^\omega \notin L_\lambda$) then $\ulcorner t \urcorner$ is the path in the computation tree going from the root to the node t^ω.

Proof. This is the counterpart of another result proved by Ong in the paper where he introduces the theory of traversals [Ong06b, proposition 6]. The original proof—an induction on the traversal rules—can be adapted to take into account the constant rules and the presence of value-leaves in the traversal. We detail the case (Lam) only. We need to show that n's binder occurs only once in the P-view at that point. By the induction hypothesis (ii) we have that $\ulcorner t \cdot \lambda\overline{\xi} \urcorner$ is a path in the computation tree from the root to $\lambda\overline{\xi}$. But n's binder occurs only once in this path, therefore the traversal $t \cdot \lambda\overline{\xi} \cdot n$ is well-defined and satisfies P-visibility. Thus (i) is satisfied. Furthermore n is a child of $\lambda\overline{\xi}$ therefore (ii) also holds. ☐

Lemma 4.30. *If $t \cdot n$ is a traversal with $n \in IN_{\mathsf{var}} \cup IN_\Sigma \cup IN_@$ then $t \neq \epsilon$ and t^ω is n's parent in $\tau(M)$ (and is thus a lambda node).*

Proof. By inspecting the traversal rules, we observe that (Lam) is the only rule which can visit a node in $IN_{\mathsf{var}} \cup IN_\Sigma \cup IN_@$. Hence t is not empty and t^ω is n's parent in $\tau(M)$. ☐

Lemma 4.31. *Suppose that M is β-normal. Let t be a traversal of $\tau(M)$ and n be a node occurring in t. Then the root $\circledast$ does not hereditarily enable n if and only if n is hereditarily enabled by some node in IN_Σ. Formally:*

$$n \notin IN^{\circledast\vdash} \quad \iff \quad n \in IN^{IN_\Sigma \vdash} \ .$$

Proof. In a computation tree, the only nodes that do not have justification pointer are: the root $\circledast$, @-nodes and Σ-constant nodes. But since M is in β-normal form, there is no @-node in the computation tree. Hence nodes are either hereditarily enabled by $\circledast$ or hereditarily enabled by some node in IN_Σ. Moreover $\circledast$ is not in IN_Σ therefore the "or" is exclusive: a node cannot be both hereditarily enabled by $\circledast$ and by some node in IN_Σ. ☐

Lemma 4.32 (The O-view is contained in a single thread)**.** *Let $t \in \mathcal{T}rav(M)$.*

(a) If $t = \ldots \cdot m \cdot n$ where m is a P-node and n is an O-node then m and n are in the same thread in t: they are hereditarily justified by the same initial occurrence (which is either $\tau(M)$'s root, a Σ-constant or an @-node);

(b) All the nodes in $\llcorner t \lrcorner$ belong to the same thread.

Proof. Clearly (b) follows immediately from (a) due to the way the O-view is computed. We show (a) by induction on the last traversal rule used to form t. The results trivially hold for the base cases (Empty) and (Root). Step case: Take $t = t' \cdot n$. If $n \in IN_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$ then we do not need to show (a). Otherwise n is an O-node. By O-visibility, n points in $\llcorner t' \lrcorner$, thus by the I.H. it must belong to the same thread as all the nodes in $\llcorner t' \lrcorner$ and in particular to the thread of t'^ω. Therefore both (i) and (ii) hold. ☐

4.1.3.4 Traversal core

Occurrences of input-variable nodes correspond to points in the computation where the term interacts with its context. At these points, a traversal can be extended in a non-deterministic way. In contrast, at a node that is hereditarily enabled by an @-node or by a constant node, the next visited node is uniquely determined. We can therefore think of such nodes as being "internal" to the computation: their semantics is predetermined and cannot be altered by the context in which the term appears. If we want to extract the essence of the computation from a traversal, a natural way to proceed thus consists in keeping only occurrences of nodes that are hereditarily enabled by the root:

Definition 4.33. The ***core of a traversal*** t, written $t \upharpoonright \circledast$, is defined as $t \upharpoonright N^{\circledast\vdash}$ (*i.e.*, the subsequence of t consisting of the occurrences of nodes that are hereditarily enabled by the root $\circledast$ of the computation tree). The set of traversal cores of M is denoted by $\mathcal{T}rav(M)^{\upharpoonright\circledast}$:

$$\mathcal{T}rav(M)^{\upharpoonright\circledast} \stackrel{\text{def}}{=} \{t \upharpoonright \circledast \ : \ t \in \mathcal{T}rav(M)\} \ .$$

Example 4.34. The core of the traversal given in example 4.20 is:

$$t \upharpoonright \lambda f z = \lambda f z \cdot f \cdot \lambda \cdot z \ .$$

Remark 4.35

- The root occurs at most once in a traversal, therefore if t is a non-empty traversal then its core is given by $t \upharpoonright r$ where r denotes the only occurrence of $\circledast$ in t. Thus we have:

$$\mathcal{T}rav(M)^{\upharpoonright\circledast} = \{t \upharpoonright r \ : \ t \in \mathcal{T}rav(M) \text{ and } r \text{ is the only occurrence of } \circledast \text{ in } t\} \ .$$

- Since @-nodes and Σ-constants do not have pointers, the traversal cores contains only nodes in $N_\lambda \cup N_{\mathsf{var}}$.

4.1.3.5 Removing @-nodes and Σ-nodes from traversals

Application nodes are essential in the definition of computation trees: they are necessary to connect together the operator and operands of an application. They also have another advantage: they ensure that the lambda-nodes are all at even level in the computation tree, which subsequently guarantees that traversals respect a certain form of alternation between lambda nodes and non-lambda nodes. Application nodes are however redundant in the sense that they do not play any role in the computation of the term. In fact it will be necessary to filter them out in order to establish the correspondence with game semantics.

Definition 4.36 (@-free traversal)**.** Let t be a traversal of $\tau(M)$. We write $t - @$ for the sequence of nodes-with-pointers obtained by

- removing from t all occurrences of @-nodes and their children value-leaves;
- replacing any link pointing to an @-node by a link pointing to the immediate predecessor of @ in t.

Suppose $u = t - @$ is a sequence of nodes obtained by applying the previously defined transformation on the traversal t, then t can be partially recovered from u by reinserting the @-nodes as follows. For each @-node in the computation tree with parent node denoted by p, we perform the following operations:

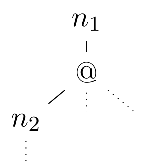
1. replace every occurrence of the pattern $p \cdot n$ for some λ-node n, by $p \cdot @ \cdot n$;

2. replace any link in u starting from a λ-node and pointing to p by a link pointing to the inserted @-node;

3. for each occurrence in u of a value-leaf v_p pointing to p, insert the value-leaf $v_@$ immediately before v_p and make it point to the immediate successor of p (which is precisely the @-node inserted in step 1.).

We write $u + @$ for this second transformation.

These transformations are well-defined because in a traversal, an @-node is always immediately preceded by its parent node n_1, and immediately followed by its first child n_2:



Example 4.37. Let f be a Σ-constant and $t = \lambda\overline{\xi} \cdot @ \cdot \lambda x \cdot f \cdot \lambda \cdot x$. Then

$$t - @ = \lambda\overline{\xi} \cdot \lambda x \cdot f \cdot \lambda \cdot x \ .$$

Example 4.38. Let t be the traversal given in example 4.20, we have:

$$t - @ = \lambda f z \cdot \lambda uv \cdot u \cdot \lambda y\ f \cdot \lambda \cdot y \cdot \lambda \cdot v \cdot \lambda \cdot z \ .$$

We also want to remove Σ-nodes from the traversals. To that end we define the operation $-\Sigma$ and $+\Sigma$ in the exact same way as $-@$ and $+@$. Again these transformations are well-defined since in a traversal, a Σ-node f is always immediately preceded by its parent node p, and a value-node v_p is always immediately preceded by a value-node v_f.

Note that the operations $-@$ and $-\Sigma$ are commutative: $(t - @) - \Sigma = (t - \Sigma) - @$.

Lemma 4.39. *For every non-empty traversal $t = t' \cdot t^\omega$ in $\mathcal{T}rav(M)$:*

$$(t - @) + @ = \begin{cases} t, & \text{if } t^\omega \notin N_@ \ ; \\ t', & \text{if } t^\omega \in N_@ \ ; \end{cases}$$

$$(t - \Sigma) + \Sigma = \begin{cases} t, & \text{if } t^\omega \notin N_\Sigma \ ; \\ t', & \text{if } t^\omega \in N_\Sigma \ . \end{cases}$$

Proof. The result follows immediately from the definition of the operation -@ and +@ (resp. $-\Sigma$ and $+\Sigma$). ∎

Remark 4.40 Sequences of the form $t - @$ (resp. $t - \Sigma$) are not, strictly speaking, proper justified sequences of nodes since after removing @-nodes, all the prime λ-nodes become justified by their parent's parent which are also λ-nodes! Moreover, these sequences do not respect alternation since two λ-nodes may become adjacent after removing a @-node.

We write $t^\star$ to denote the sequence obtained from t by removing all the @-nodes as well as the constant nodes together with their associated value-leaves:

$$t^\star \stackrel{\text{def}}{=} t - @ - \Sigma \ .$$

Example 4.41. Let f be a Σ-constant. We have

$$\left(\lambda\overline{\xi} \cdot @ \cdot \lambda x \cdot f \cdot \lambda \cdot x\right)^\star = \lambda\overline{\xi} \cdot \lambda x \cdot \lambda \cdot x \ .$$

We introduce the set
$$\mathcal{T}rav(M)^\star = \{t^\star \mid t \in \mathcal{T}rav(M)\} \ .$$

Remark 4.42 If M is a β-normal term and if it contains no Σ-constant (as for pure simply-typed terms) then $\tau(M)$ does not contain any @-node or Σ-node, thus all nodes are hereditarily enabled by $\circledast$ and we have $\mathcal{T}rav(M) = \mathcal{T}rav(M)^{\upharpoonright \circledast} = \mathcal{T}rav(M)^\star$.

Lemma 4.43. *For every traversal t we have $t^\star \upharpoonright N^{\circledast\vdash} = t \upharpoonright \circledast$.*

Proof. This is because nodes removed by the operation $\_^\star$ are not hereditarily enabled by the root of the tree. ∎

The notion of P-view extends naturally to sequences of the form $t^\star$: it is defined by the same induction as for P-views of traversals. It is then easy to check that if t^ω is not in $L_@ \cup L_\Sigma$ then the P-view of $t^\star$ is obtained from $\ulcorner t \urcorner$ by keeping only the non @/Σ-nodes:

$$\ulcorner t^\star \urcorner = \ulcorner t \urcorner \setminus (N_@ \cup N_\Sigma) \ . \tag{4.1}$$

We define a projection operation for sequences of the form $t^\star$ as follows:

Definition 4.44. Let t be a traversal such that $t^\omega \notin L_@ \cup L_\Sigma$ and $n \in N_\lambda$ be a lambda-node. The projection $t^\star \upharpoonright N^{(n)}$ is defined as the subsequence of $t^\star$ consisting of nodes of $N^{(n)}$ only. If a variable node loses its pointer in $t^\star \upharpoonright N^{(n)}$ then its justifier is reassigned to the only occurrence of n in $\ulcorner t^\star \urcorner$.

Note that this operation is well-defined. Indeed if a variable x loses its pointer in $t^\star \upharpoonright N^{(n)}$ then it means that x is free in $M^{(n)}$. But then n must occur in the path to the root $\circledast$ which is precisely $\ulcorner t_{\leqslant x} \urcorner$. Thus by (4.1), n must occur in $\ulcorner {t_{\leqslant x}}^\star \urcorner$.

4.1.3.6 Subterm projection (with respect to a node occurrence)

Let n be a node-occurrence in a traversal t. The ***subterm projection*** $t \upharpoonright\!\upharpoonright n$ is defined as the subsequence of t consisting of the occurrences whose P-view at that point contain the node n. Formally:

Definition 4.45. Let $t \in \mathcal{T}rav(M)$ and n be an occurrence in t. The subsequence $t \upharpoonright\!\upharpoonright n$ of t is defined inductively on t as follows:

- $(t \cdot n) \upharpoonright\!\upharpoonright n = n$;
- If m is an O-node and $m \neq n$ then
$$(t \cdot m) \upharpoonright\!\upharpoonright n = \begin{cases} (t \upharpoonright\!\upharpoonright n) \cdot m, & \text{if } m\text{'s justifier appears in } t \upharpoonright\!\upharpoonright n \ ; \\ t \upharpoonright\!\upharpoonright n, & \text{otherwise ;} \end{cases}$$
- If m is a P-node and $m \neq n$ then
$$(t \cdot m) \upharpoonright\!\upharpoonright n = \begin{cases} (t \upharpoonright\!\upharpoonright n) \cdot m, & \text{if } t^\omega\text{'s appears in } t \upharpoonright\!\upharpoonright n \ ; \\ t \upharpoonright\!\upharpoonright n, & \text{otherwise ;} \end{cases}$$

  where in the first subcase, if m loses its justifier in $t \upharpoonright\!\upharpoonright n$ then it is reassigned to n.

We call this transformation the *subterm projection with respect to a node occurrence* because it keeps only nodes that appear in the sub-tree rooted at some reference node. If n is an occurrence of a lambda node $\lambda\overline{\xi} \in IN_\lambda$ then we say that $t \upharpoonright\!\upharpoonright n$ a ***sub-traversal of the computation tree*** $\tau(M)$. This name is suggestive of the forthcoming Proposition 4.62 stating that $t \upharpoonright\!\upharpoonright n$ is a traversal of the sub-computation tree of $\tau(M)$ rooted at $\lambda\overline{\xi}$.

REMARK 4.46 There is an alternative way to define $t \upharpoonright\upharpoonright r_0$: For every traversal t we write t^+ to denote the sequence-with-pointers obtained from t by adding pointers as follows: For every occurrence of a @ or Σ-node m in t we add a pointer going from m to its predecessor in t (which is necessarily an occurrence of its parent node). Further, for every variable node x we add auxiliary pointers going to each lambda node occurring in the P-view at that point after x's binder. Conversely, for every sequence-with-pointers u we define u^- as the sequence obtained from u by removing the links associated to @ and Σ-nodes and where for each occurrence of a variable node, only the "longest" link is preserved. (The length of a link being defined as the distance between the source and the target occurrence.) Clearly the operation $\_^-$ is the inverse of $\_^+$: For every traversal t we have $t = (t^+)^-$. Then it can be easily shown that the sequence $t \upharpoonright\upharpoonright n$ is precisely the subsequence of t consisting of nodes hereditarily justified by n *with respect to the justification pointers of* t^+:

$$t \upharpoonright\upharpoonright n = (t^+ \upharpoonright n)^- \ .$$

(Note that since the operation $\_^+$ changes the justification pointers, the hereditary justification relation in a traversal t is different from the hereditary justification relation in t^+ and therefore we have $(t \upharpoonright n)^+ \sqsubseteq t^+ \upharpoonright n$ but $(t \upharpoonright n)^+ \neq t^+ \upharpoonright n$.) End of remark.

The following lemmas follow directly from the definition of the subterm projection $t \upharpoonright\upharpoonright n$:

Lemma 4.47. *Let t be a traversal and r be a lambda node occurring in t.*

(a) Suppose that $t = \ldots m \ldots n$ where n is an O-node and $n \neq r$. Then n appears in $t \upharpoonright\upharpoonright r$ if and only if m appears in $t \upharpoonright\upharpoonright r$.

(b) Suppose that $t = \ldots \cdot n$ where n is a P-node. Then n appears in $t \upharpoonright\upharpoonright r$ if and only if the last lambda node in $\ulcorner t \urcorner$ does.

(c) Suppose that $t = \ldots m \ldots v_m$ for some leaf-node $v_m \in L$. Then v_m appears in $t \upharpoonright\upharpoonright r$ if and only if m does.

Proof. (a) holds by definition of $t \upharpoonright\upharpoonright r$. (b) is proved by induction on t: It follows easily from the fact that in the definition of $t \upharpoonright\upharpoonright r$, the inductive cases follow those from the definition of traversal P-views. (c) If $v_m \in L_@ \cup L_\Sigma \cup L_{\mathsf{var}}$ then it falls back to (a). Otherwise $v_m \in L_\lambda$ and by (b), v_m appears in $t \upharpoonright\upharpoonright r$ if and only if the last lambda node in $\ulcorner t \urcorner$ does. But the last node in $\ulcorner t \urcorner$ is necessarily m (since v_m is necessarily visited using a copy-cat rule). ☐

Lemma 4.48. *Let $t \in \mathcal{T}rav(M)$ and r be the occurrence in t of a λ-node. We have:*

$$?(t \upharpoonright\upharpoonright r) = ?(t) \upharpoonright\upharpoonright r \ .$$

Proof. Take a prefix u of t ending with a value-leaf v_n of an occurrence n. By Lemma 4.47(c), the operation $\_ \upharpoonright\upharpoonright r$ removes v_n from t if and only if it also removes n. ☐

4.1.3.7 O-view and P-view of the subterm projection

P-view projection

Lemma 4.49 (P-view projection for traversals)**.** *Let t be a traversal and r_0 be an occurrence in t of a lambda node $r \in IN_\lambda$. Then:*

(i) If t^ω appears in $t \upharpoonright\upharpoonright r_0$ then:

a. r_0 appears in $\ulcorner t \urcorner$, all the nodes occurring after r_0 in $\ulcorner t \urcorner$ appear in $t \upharpoonright\upharpoonright r_0$ and all the nodes occurring before r_0 in $\ulcorner t \urcorner$ do not appear in $t \upharpoonright\upharpoonright r_0$;

b. $\ulcorner t \upharpoonright\upharpoonright r_0 \urcorner^{M(r)} = \ulcorner t \urcorner^M_{\geqslant r_0} = r_0 \cdot \ldots$;

c. *if* t^ω *also appears in* $t \upharpoonright\!\upharpoonright r_1$ *for some occurrence* r_1 *of* r *then* $r_0 = r_1$*;*

d. *if* $t = \ldots \overset{\curvearrowleft}{m \ldots n}$ *and* m *does not appear in* $t \upharpoonright\!\upharpoonright r_0$ *then* r_0 *occurs after* m *in* t *and* m *is a free variable node in the sub-computation tree* $\tau(M^{(r)})$*.*

(ii) Suppose $t = \ldots r_0 \ldots \overset{\curvearrowleft}{m \ldots n}$*. Then the node* n *appears in* $t \upharpoonright\!\upharpoonright r_0$ *if and only if* m *does.*

Proof. (i) A trivial induction shows both a. and b. (The inductive steps in the definition of the projection operation $\_ \upharpoonright\!\upharpoonright r_0$ correspond precisely to those from the definition of P-views.)

c. By a., both r_0 and r_1 appears in the P-view. But the P-view is the path from t^ω to the root, hence it cannot contain two different occurrences of the same node r.

d. Since t^ω appears in $t \upharpoonright\!\upharpoonright r_0$ and its justifier m is not in $t \upharpoonright\!\upharpoonright r_0$, by a., the justifier m necessarily precedes r_0 in t, and by Lemma 4.47, n is necessarily a variable node. Thus m occurs before r_0 in the P-view $\ulcorner t \urcorner$. In other words, r_0 lies in the path from n to its binder m. Consequently, n is a free variable node in $\tau(M^{(r)})$.

(ii) The cases where n is an O-node or a leaf-node are handled by Lemma 4.47(a) and (c) respectively. Otherwise, since n is not initial it does not belong to $IN_@ \cup IN_\Sigma$ therefore $n \in IN_{\mathsf{var}}$. If n appears in $t \upharpoonright\!\upharpoonright r_0$ then by (i) all the nodes occurring in $\ulcorner t \urcorner$ up to r_0 appear in $t \upharpoonright\!\upharpoonright r_0$. By P-visibility, m appears in $\ulcorner t \urcorner$ and since r_0 precedes it by assumption, m also appears in $t \upharpoonright\!\upharpoonright r_0$. Conversely, if m appears in $t \upharpoonright\!\upharpoonright r_0$ then since m appears in the P-view at x, by definition of $t \upharpoonright\!\upharpoonright r_0$, x must also appear in $t \upharpoonright\!\upharpoonright r_0$. $\square$

Lemma 4.50. *Let* $t \in \mathcal{T}rav(M)$ *such that* $t^\omega \notin L_\lambda$*. Let* r *be some lambda node in* IN_λ*.*

The node t^ω *belongs to the subtree of* $\tau(M)$ *rooted at* r *(i.e.,* $t^\omega \in N^{(r)}$*) if and only if* t^ω *appears in* $t \upharpoonright\!\upharpoonright r_0$ *for some occurrence* r_0 *of* r *in* t*.*

Proof. Only if part: Since t's last move in not a lambda leaf, by Proposition 4.29, the P-view $\ulcorner t \urcorner$ is the path to the root $\circledast$. Hence since t^ω belongs to the subtree of $\tau(M)$ rooted at r, $\ulcorner t \urcorner$ must contain (exactly) one occurrence r_0 of r. But then by definition of $t \upharpoonright\!\upharpoonright r_0$, all the nodes following r_0 occurring in the P-view must also belong to $t \upharpoonright\!\upharpoonright r_0$, so in particular, t^ω does.

If part: By Lemma 4.49(i), r_0 must occur in $\ulcorner t \urcorner$ and therefore r_0 lies in the path from t^ω to the root $\circledast$ of the computation tree $\tau(M)$. Consequently, t^ω necessarily belongs to the subtree of $\tau(M)$ rooted at r. $\square$

Lemma 4.51. *Let* t *be a traversal and* r_0 *be an occurrence in* t *of some lambda node* r*. Then an occurrence* $n \notin N_@ \cup N_\Sigma$ *of* t *is hereditarily justified by* r_0 *in* $t^\star \upharpoonright N^{(r)}$ *if and only if* n *appears in* $t \upharpoonright\!\upharpoonright r_0$*.*

Proof. We proceed by induction on $t_{\leqslant n}$. If $n = r_0$ or if r_0 does not occur in $t_{\leqslant n}$ then the result holds trivially. Suppose that r_0 occurs in $t_{<n}$. Let m be n's justifier in t. We do a case analysis on n. The case $n \in L_@ \cup L_\Sigma \cup IN_@ \cup IN_\Sigma$ is excluded by assumption.

Suppose $n \in L_\lambda \cup L_{\mathsf{var}} \cup IN_\lambda$ then

$$\begin{array}{rcll}
n \text{ appears in } t \upharpoonright\!\upharpoonright r_0 & \iff & m \text{ appears in } t \upharpoonright\!\upharpoonright r_0 & \text{by Lemma 4.47(a)} \\
& \iff & m \text{ her. just. by } r_0 \text{ in } t^\star \upharpoonright N^{(r)} & \text{by I.H. on } t_{\leqslant m} \\
& \iff & n \text{ her. just. by } r_0 \text{ in } t^\star \upharpoonright N^{(r)} & \text{since } m \text{ is } n\text{'s parent in } \tau(M^{(r)}).
\end{array}$$

Suppose that $n \in IN_{\mathsf{var}}$ then

$$\begin{array}{rcll}
n \text{ appears in } t \upharpoonright\!\upharpoonright r_0 & \iff & r_0 \text{ appears in } \ulcorner t \urcorner & \text{by Lemma 4.50 and 4.49(i)} \\
& \iff & \left\{ \begin{array}{l} r_0 \text{ precedes } m \text{ in } \ulcorner t \urcorner, \text{ and thus } n \text{ is a bound variable in } M^{(r)} \\ \text{or } r_0 \text{ appears strictly after } m \text{ in } \ulcorner t \urcorner \text{ and } n \text{ is free in } M^{(r)} \end{array} \right. & \\
& \iff & \left\{ \begin{array}{ll} m \text{ appears in } t \upharpoonright\!\upharpoonright r_0 & \text{by Lemma 4.49(i)} \\ \text{or } n \text{ points to } r_0 \text{ in } t^\star \upharpoonright N^{(r)} & \text{by def. of } _ \upharpoonright N^{(r)} \end{array} \right. &
\end{array}$$

$$\begin{aligned}
&\iff \left\{ \begin{array}{l} m \text{ her. just. by } r_0 \text{ in } t^\star \upharpoonright N^{(r)} \\ \text{or } n \text{ points to } r_0 \text{ in } t^\star \upharpoonright N^{(r)} \end{array} \right. && \text{by I.H. on } t_{\leqslant m} \\
&\iff \left\{ \begin{array}{l} n \text{ her. just. by } r_0 \text{ in } t^\star \upharpoonright N^{(r)} \\ \text{or } n \text{ points to } r_0 \text{ in } t^\star \upharpoonright N^{(r)} \end{array} \right. && n \text{ is in } N^{(r)} \text{ iff its binder } m \text{ is} \\
&\iff n \text{ is her. just. by } r_0 \text{ in } t^\star \upharpoonright N^{(r)} \quad .
\end{aligned}$$

□

Lemma 4.52. *Take a traversal t. Let r be a node in IN_λ and r_0 an occurrence of r in t. Suppose that t^ω appears in $t \upharpoonright\upharpoonright r_0$ and that the thread of t^ω is initiated by $\alpha \in IN_@ \cup IN_\Sigma$.*

(i) If r_0 precedes α in t then all the nodes occurring in the thread appear in $t \upharpoonright\upharpoonright r_0$.

(ii) If α precedes r_0 in t then t^ω is hereditarily enabled by r in $\tau(M^{(r)})$.

Proof. (i) By definition of a thread, the nodes occurring in the thread are all hereditarily justified by α. Since r_0 precedes α and t^ω appears in $t \upharpoonright\upharpoonright r_0$, by Lemma 4.49(ii) all the nodes in the thread must also appear in $t \upharpoonright\upharpoonright r_0$.

(ii) Let q be the first node in t that hereditarily justifies t^ω in t and appears in $t \upharpoonright\upharpoonright r_0$. There are three cases. (i) If $q \in IN_\lambda$ then necessarily $q = r_0$. Otherwise by definition of $\_ \upharpoonright\upharpoonright r_0$, q's justifier also appears in $t \upharpoonright\upharpoonright r_0$ which contradicts the definition of q. Hence the result holds trivially. (ii) If $q \in IN_@ \cup IN_\Sigma$ then necessarily $q = \alpha$, since links always point inside the current thread and since a thread contains by definition only one node in $IN_@ \cup IN_\Sigma$. But α precedes r_0 therefore α cannot be hereditarily justified by r_0 hence this case is not possible. (iii) If $q \in IN_{\mathsf{var}}$ then by Lemma 4.49(i.d), q is an free variable in $\tau(M^{(r)})$ and therefore it is enabled by r in $\tau(M^{(r)})$. Hence since t^ω is hereditarily justified by r_0, it must be hereditarily enabled by r in $\tau(M^{(r)})$. □

O-view projection We now spend the rest of this section proving the following result:

Proposition 4.53 (O-view projection for traversals)**.** *Let t be a traversal of $\mathcal{T}rav(M)$ such that its last node appears in $t \upharpoonright\upharpoonright r_0$ for some occurrence r_0 in t of a lambda node r in IN_λ. Then $\llcorner t \lrcorner_M \upharpoonright\upharpoonright r_0 \sqsubseteq \llcorner t \upharpoonright\upharpoonright r_0 \lrcorner_{M^{(r)}}$.*

The reader may recognize the similarity with another result of game semantics from the seminal paper by Hyland and Ong on full abstraction of PCF [HO00]:

Proposition 4.54 (P-view projection in game semantics)**.** *[HO00, Prop.4.3] Let s be a legal position of a game $A \to B$. If s^ω is in B then $\ulcorner s \urcorner^{A \to B} \upharpoonright B \sqsubseteq \ulcorner s \upharpoonright B \urcorner^B$.*

The proof of this proposition is non-trivial, so we can expect a proof of Proposition 4.53 to be just as hard. Instead of deriving an equivalently complicated proof we proceed by establishing an analogy between the two settings, so that the proof of Proposition 4.54 maps, by this analogy, to a proof of Proposition 4.53.

The idea of the analogy is simple: justified sequences of moves correspond to justified sequences of nodes. We now need to make sure that the assumptions used in the proof of Proposition 4.54 are also valid in the traversal setting. One assumption is that the position s is *legal*:

- (w1) Initial question to start: The first move played in s is an initial move and there is no other occurrence of initial moves in the rest of s;
- (w2) Alternation: P-moves and O-moves alternate in s;
- (w3) Explicit justification: *every* move, except the first one, has a pointer to a preceding move,
- (w4) Well-bracketing: The pending question is answered first;
- (w5) Visibility: s satisfies P-visibility and O-visibility.

Further the proof makes use of some properties implied by the fact that s is a *position* of the game $A \to B$:

- (w6) For every occurrence n in the position, $n \in A \iff n \notin B$;
- (w7) Switching condition: The Proponent is the only player who can switch from game A to B or from B to A.
- (w8) Justification in $A \to B$: Suppose m justifies n in s. Then
  - $n \in B$ implies $m \in B$;
  - if n is a non-initial move in A then $n \in A$;
  - if n is an initial move in A then $n \in B$.

A close inspection shows that those are the only assumptions used in the proof of Proposition 4.54. Most of these requirements coincide with properties that we have already shown for traversals, but traversals do not strictly satisfy explicit justification (w3): there are nodes—the @-nodes and Σ-nodes—that do not have justification pointers. The solution to this problem is simple: we just add justification pointers to @-nodes and Σ-nodes!

Take a justified sequence of nodes t. We define $\mathsf{ext}(t)$, the ***extension of*** t, to be the sequence of nodes-with-pointers obtained from $\diamond \cdot t$ (where $\diamond$ is a dummy node) by adding justification pointers going from occurrences of the root $\circledast$, @-nodes and Σ-nodes to their immediate predecessor in t.

Example 4.55. Let $f \in \Sigma$. We have $\mathsf{ext}(\lambda\overline{\xi} \cdot @ \cdot \lambda x \cdot f \cdot \lambda \cdot x) = \diamond \cdot \lambda\overline{\xi} \cdot @ \cdot \lambda x \cdot f \cdot \lambda \cdot x$.

It is an immediate fact that for every two justified sequences t_1 and t_2 we have:

$$\mathsf{ext}(t_1) \sqsubseteq \mathsf{ext}(t_2) \qquad \iff \qquad t_1 \sqsubseteq t_2 \tag{4.2}$$

and for every justified sequence t:

$$\mathsf{ext}(t) \upharpoonright\!\upharpoonright r_0 = \mathsf{ext}(t \upharpoonright\!\upharpoonright r_0) \ . \tag{4.3}$$

Since a traversal extension $\mathsf{ext}(t)$ may contain @/Σ-nodes with pointers, it is not a proper justified sequence of nodes as defined in Def. 4.11. Nevertheless, the basic transformations that we have defined for justified sequences—such as hereditary projection, P-view and O-view—apply naturally to traversal extensions (without any modification in their definition). The views of a traversal extension can be expressed in term of the traversal's views as follows:

$$\llcorner \mathsf{ext}(t) \lrcorner = \llcorner t \lrcorner \tag{4.4}$$

$$\ulcorner \mathsf{ext}(t) \urcorner = \begin{cases} \epsilon, & \text{if } t = \epsilon \ ; \\ \diamond \cdot \mathsf{ext}(\ulcorner t \urcorner), & \text{otherwise.} \end{cases} \tag{4.5}$$

The transformations $\ulcorner \cdot \urcorner$ and $\llcorner \cdot \lrcorner$, however, do not convey the appropriate notions of view for extended traversals. We thus define more appropriate notions of view for traversal extensions, called O-e-view and P-e-view, as follows:

Definition 4.56. The O-e-view of a traversal extension $\mathsf{ext}(t)$, written, $\llcorner \mathsf{ext}(t) \lrcorner_{\mathsf{e}}$ is defined as

$$\llcorner \mathsf{ext}(t) \lrcorner_{\mathsf{e}} \stackrel{\mathrm{def}}{=} \ulcorner \mathsf{ext}(t) \urcorner \ .$$

The P-e-view of $\mathsf{ext}(t)$, written, $\llcorner \mathsf{ext}(t) \lrcorner_{\mathsf{e}}$ is defined by induction:

$$\begin{array}{rcll} \ulcorner \epsilon \urcorner^{\mathsf{e}} & = & \epsilon & \\ \ulcorner u \cdot n \urcorner^{\mathsf{e}} & = & \ulcorner u \urcorner^{\mathsf{e}} \cdot n & \text{if } n \text{ is an O-node;} \\ \ulcorner u \cdot m \cdot \ldots \cdot n \urcorner^{\mathsf{e}} & = & \ulcorner u \urcorner^{\mathsf{e}} \cdot m \cdot n & \text{if } n \text{ is a P-node.} \end{array}$$

Inserting a dummy node $\diamond$ at the beginning of the traversal changes the parity of the alternation between P-nodes and O-nodes. Thus the role of O and P is interchanged for traversal extensions. This explains why the O-e-view is calculated from the P-view.

For the P-e-view, the definition is almost the same as the traversal O-view $\llcorner \cdot \lrcorner$ except that the computation does not stop when reaching a node in $IN_{@} \cup IN_{\Sigma}$—this is sometimes referred as the *long O-view* [Har05]. (The O-view contains only one thread whereas the long-O-view may contain several; the O-view is a suffix of the long O-view.) This is possible because occurrences of nodes from $IN_{@} \cup IN_{\Sigma}$ in a traversal extension all have a justification pointer. The O-view of t is a suffix of its P-e-view:

$$\ulcorner t \urcorner^{\mathsf{e}} = w \cdot \llcorner t \lrcorner \quad \text{for some sequence } w. \tag{4.6}$$

We are now fully equipped to establish an analogy between the traversal extension setting and the game-semantic setting. The reason why we make this analogy is purely to reuse the proof of Proposition 4.54 [HO00, Prop. 4.3]. The reader must not confuse it with another correspondence that we will establish in a forthcoming section, between plays of game semantics and traversals of the computation tree. (In particular the Proponent/Opponent colouring used here is the opposite of the one used in the Correspondence Theorem.) The following analogy is made:

Traversal setting	**Game-semantic setting**
Extended traversal $\mathsf{ext}(t)$	Play s
P-Nodes $(IN_{\mathsf{var}} \cup IN_{\Sigma} \cup IN_{@} \cup L_{\lambda})$ or $\diamond$	O-moves $\bullet$
O-nodes $(L_{\mathsf{var}} \cup L_{\Sigma} \cup L_{@} \cup IN_{\lambda})$	P-moves $\circ$
P-view $\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}}$	P-view $\ulcorner s \urcorner$
O-view $\llcorner \mathsf{ext}(t) \lrcorner_{\mathsf{e}}$	O-view $\llcorner s \lrcorner$
Occurrence n appearing in $t \upharpoonright\upharpoonright r_0$	Occurrence $n \in B$
Occurrence n not appearing in $t \upharpoonright\upharpoonright r_0$	Occurrence $n \in A$
No notion of initiality (All nodes are considered to be non-initial).	Distinction between initial and non-initial move.

Clearly sequences of the form $\mathsf{ext}(t)$ satisfy the requirements (w1) to (w5): For (w1), the initial node becomes $\diamond$. Explicit justification (w4) holds since we have added pointers to $@/\Sigma$-nodes. Finally, alternation (w3), well-bracketing (w4) and visibility (w5) of the traversal t (Prop. 4.29) are preserved by the extension operation (where visibility is defined with respect to the appropriate notion of P-view and O-view).

The property (w6) trivially holds: $n \in t \upharpoonright\upharpoonright r_0$ iff $\neg(n \notin t \upharpoonright\upharpoonright r_0)$. So does the switching condition (w7): if $t = \ldots \cdot m \cdot n$ where n is a P-node and m is an O-node then, by definition of $t \upharpoonright\upharpoonright r_0$, m appears in $t \upharpoonright\upharpoonright r_0$ if and only if n does. For (w8): Using the analogy of the preceding table and since all nodes are considered "non-initial" in $\mathsf{ext}(t)$, this condition can be stated as:

(w8) Suppose m justifies n in $\mathsf{ext}(t)$. Then $n \in t \upharpoonright\upharpoonright r_0$ if and only if $m \in t \upharpoonright\upharpoonright r_0$.

Unfortunately, as we have seen previously, the direct implication does not hold in general! (Indeed, a variable node can very well appear in $t \upharpoonright\upharpoonright r_0$ even though its justifier does not.) Consequently, the proof of Proposition 4.54 cannot be directly reused in our setting. A weaker version of condition (w8) holds however: if r_0 occurs before n's justifier then, by Lemma 4.49(i), n appears in $t \upharpoonright\upharpoonright r_0$ if and only if its justifier does; this condition turns out to be sufficient to reuse most of the proof of Proposition 4.54 [HO00].

We reproduce here some definition used in this proof. Let s be a position of the game $A \rightarrow B$. A bounded segment is a segment θ of s of the form $\overset{x}{\circ} \ldots \overset{y}{\bullet}$. If x is in A, and hence so does y, then θ is an A-bounded segment. Respectively if x and y are in B then it is a B-bounded segment. By an abuse of notation we define $\ulcorner \theta \upharpoonright B \urcorner$ to be the subsequence of $\ulcorner s_{\leqslant y} \upharpoonright B \urcorner$ consisting only of moves in θ appearing after (and not including) x. We then have:

Lemma 4.57. *[HO00, Lemma A.3] Let θ be an A-bounded segment in s with end-moves x and y.*

(i) $\ulcorner \theta \upharpoonright B \urcorner = \overset{p_r}{\circ} \cdot \overset{q_r}{\bullet} \ldots \overset{p_1}{\circ} \cdot \overset{q_1}{\bullet}$ *for some $r \geq 0$. Note that each segment $p_i \ldots q_i$ is B-bounded in s, for $1 \leq i \leq r$.*

(ii) For every P-move m in θ which appears in $\llcorner s_{<y} \lrcorner$, m does not belong to any of the B-bounded segments $p_i \ldots q_i$ for $1 \leq i \leq r$.

This lemma assumes that the segment θ satisfies the assumptions (w1) to (w8). As we have seen, (w8) does not always hold for extended traversals. But using our analogy with extended traversals, a segment θ is "A-bounded" if θ is bounded by two nodes appearing in $t \upharpoonright\upharpoonright r_0$. This can only happen if r_0 occurs before θ in t or if θ's left bound is r_0. Thus the condition (w8) holds at least for the nodes of the segment θ. The previous lemma thus translates into:

Lemma 4.58. *Let t be a traversal and θ be a segment of $\mathsf{ext}(t)$ bounded by nodes x and y appearing in $t \upharpoonright\upharpoonright r_0$.*

(i) $\ulcorner \theta \upharpoonright\upharpoonright r_0 \urcorner^{\mathsf{e}} = p_r \cdot q_r \ldots p_1 \cdot q_1$ *for some $r \geq 0$ where p_i is an O-node and q_i is a P-node, for $1 \leq i \leq r$.*

(ii) For every O-node m occurring in θ and appearing in $\llcorner \mathsf{ext}(t)_{<y} \lrcorner^{\mathsf{e}}$, m does not belong to any of the segments $p_i \ldots q_i$ for $1 \leq i \leq r$.

We now show the analogue of Proposition 4.54 in the context of extended traversals:

Proposition 4.59. *Let t be a traversal and r_0 be an occurrence of some lambda node $\lambda\overline{\xi}$. If $\mathsf{ext}(t)$'s last node appears in $t \upharpoonright\upharpoonright r_0$ then $\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonright\upharpoonright r_0 \sqsubseteq \ulcorner \mathsf{ext}(t \upharpoonright\upharpoonright r_0) \urcorner^{\mathsf{e}}$.*

Proof. By (4.3) we can equivalently show that: $\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonright\upharpoonright r_0 \sqsubseteq \ulcorner \mathsf{ext}(t) \upharpoonright\upharpoonright r_0 \urcorner^{\mathsf{e}}$. By induction on the length of t. The base case is immediate. For the inductive case, we do a case analysis:

- $t = t' \cdot r_0$. We have $\mathsf{ext}(t) \upharpoonright\upharpoonright r_0 = r_0$ and $\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonright\upharpoonright r_0 = r_0 = \ulcorner \mathsf{ext}(t) \upharpoonright\upharpoonright r_0 \urcorner^{\mathsf{e}}$.
- $t = t' \cdot n$ where n is an O-node and is not the occurrence r_0.

  There are two cases.

  - Suppose that the last node in t' appears in $t \upharpoonright\upharpoonright r_0$. Then by the I.H. we have $\ulcorner \mathsf{ext}(t') \urcorner^{\mathsf{e}} \upharpoonright \upharpoonright r_0 \sqsubseteq \ulcorner \mathsf{ext}(t') \upharpoonright\upharpoonright r_0 \urcorner^{\mathsf{e}}$ thus

$$
\begin{aligned}
\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonright\upharpoonright r_0 &= \ulcorner \mathsf{ext}(t') \urcorner^{\mathsf{e}} \upharpoonright\upharpoonright r_0 \cdot n && \text{(P-view for extended justified sequences of nodes of } M) \\
&\sqsubseteq \ulcorner \mathsf{ext}(t') \upharpoonright\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot n && \text{(induction hypothesis)} \\
&= \ulcorner \mathsf{ext}(t') \upharpoonright\upharpoonright r_0 \cdot n \urcorner^{\mathsf{e}} && \text{(P-view for extended justified sequences of nodes of } M^{(\lambda\overline{\xi})}, n \text{ belongs to } N^{(\lambda\overline{\xi})} \text{ by Lemma 4.50)} \\
&= \ulcorner \mathsf{ext}(t' \cdot n) \upharpoonright\upharpoonright r_0 \urcorner^{\mathsf{e}} && (n \text{ occurs in } t \upharpoonright\upharpoonright r_0) \\
&= \ulcorner \mathsf{ext}(t) \upharpoonright\upharpoonright r_0 \urcorner^{\mathsf{e}} && \text{(definition of } t).
\end{aligned}
$$

  - Suppose that the last node y_1 in t' does not appear in $t \upharpoonright\upharpoonright r_0$. Let $\underline{m}$ be the last node preceding m in $\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}}$ that appears in $t \upharpoonright\upharpoonright r_0$. Then for some $q \geq 0$ we have

$$
\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} = \ulcorner \mathsf{ext}(t)_{\leqslant \underline{m}} \urcorner^{\mathsf{e}} \cdot \underbrace{x_q \cdot y_q \ \ldots \ x_1 \cdot y_1}_{\text{all appear in } t \upharpoonright\upharpoonright r_0 \cdot m}
$$

  where the x_is are all O-nodes and the y_is are all P-nodes.

Therefore the sequence $\mathsf{ext}(t)$ must be of the following form:

$$\mathsf{ext}(t)_{\leqslant m} \cdot \underbrace{x_q \ldots y_q}_{\theta_q} \cdots \underbrace{x_1 \cdots y_1}_{\theta_1} \cdot m$$

where each segment θ_i is bounded by nodes appearing in $t \upharpoonleft\!\upharpoonright r_0$. By Lemma 4.58, when computing the P-view of $\mathsf{ext}(t)$, pointers going from a segment θ to a node outside the segment are never followed! In other words:

$$\ulcorner \mathsf{ext}(t) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} = \ulcorner \mathsf{ext}(t)_{\leqslant m} \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot \ulcorner \theta_q \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot \ \cdots \ \cdot \ulcorner \theta_1 \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot m \ .$$

Hence:

$$\begin{aligned}
\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 &= \ulcorner \mathsf{ext}(t)_{\leqslant m} \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 \cdot n \\
&\sqsubseteq \ulcorner \mathsf{ext}(t)_{\leqslant m} \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot n && \text{(I.H.)} \\
&\sqsubseteq \ulcorner \mathsf{ext}(t)_{\leqslant m} \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot \ulcorner \theta_q \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot \ldots \cdot \ulcorner \theta_1 \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot n \\
&= \ulcorner \mathsf{ext}(t) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} && \text{(by the previous equation).}
\end{aligned}$$

- $t = t' \cdot m \cdot u \cdot n$ where n is a P-node. Then m is an O-node.

  Suppose that r_0 appears in $t' \cdot m$, then since n appears in $t \upharpoonleft\!\upharpoonright r_0$, by Lemma 4.49(i) so does m. Thus we can apply the I.H. on $t' \cdot m$:

$$\begin{aligned}
\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 &= \ulcorner \mathsf{ext}(t') \cdot m \cdot \overline{u} \cdot n \urcorner^{\mathsf{e}}_M \upharpoonleft\!\upharpoonright r_0 && \text{(definition of } t) \\
&= (\ulcorner \mathsf{ext}(t') \cdot m \urcorner^{\mathsf{e}} \cdot n) \upharpoonleft\!\upharpoonright r_0 && \text{(P-eview computation in } M\ ) \\
&= \ulcorner \mathsf{ext}(t' \cdot m) \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 \cdot n && (n \text{ appears in } t \upharpoonleft\!\upharpoonright r_0) \\
&\sqsubseteq \ulcorner (\mathsf{ext}(t' \cdot m)) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot n && \text{(induction hypothesis on } t' \cdot m) \\
&= \ulcorner \mathsf{ext}(t') \upharpoonleft\!\upharpoonright r_0 \cdot m \urcorner^{\mathsf{e}} \cdot n && (m \text{ appears in } t \upharpoonleft\!\upharpoonright r_0) \\
&= \ulcorner \mathsf{ext}(t') \upharpoonleft\!\upharpoonright r_0 \cdot m \cdot (\mathsf{ext}(u) \upharpoonleft\!\upharpoonright r_0) \cdot n \urcorner^{\mathsf{e}} && \text{(P-eview in } M^{(\lambda \overline{\xi})}\text{, nodes in} \\
& && m \cdot (\mathsf{ext}(u) \upharpoonleft\!\upharpoonright r_0) \cdot n \text{ are all in } N^{(\lambda \overline{\xi})}) \\
&= \ulcorner (\mathsf{ext}(t') \cdot m \cdot \mathsf{ext}(u) \cdot n) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} && (m \text{ and } n \text{ both appear in } t \upharpoonleft\!\upharpoonright r_0) \\
&= \ulcorner \mathsf{ext}(t) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} && \text{(definition of } t).
\end{aligned}$$

  Suppose that r_0 appears in u then:

$$\begin{aligned}
\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 &= \ulcorner \mathsf{ext}(t' \cdot m) \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 \cdot n \\
&= n && (r_0 \text{ occurs after } m) \\
&\sqsubseteq \ulcorner (\mathsf{ext}(t' \cdot m)) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot n \\
&= \ulcorner \mathsf{ext}(t) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \ . && \square
\end{aligned}$$

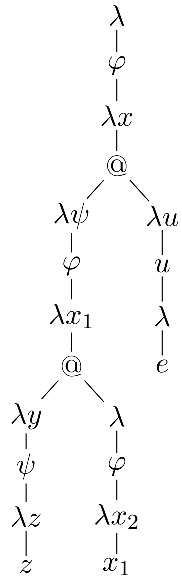
We can now prove Proposition 4.53:

Proof of Proposition 4.53. We have:

$$\begin{aligned}
\llcorner t \lrcorner \upharpoonleft\!\upharpoonright r_0 &= \llcorner \mathsf{ext}(t) \lrcorner \upharpoonleft\!\upharpoonright r_0 && \text{by (4.4)} \\
&\sqsubseteq \ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 && \text{by (4.6)} \\
&\sqsubseteq \ulcorner \mathsf{ext}(t \upharpoonleft\!\upharpoonright r_0) \urcorner^{\mathsf{e}} && \text{by Proposition 4.59} \\
&= w \cdot \llcorner \mathsf{ext}(t \upharpoonleft\!\upharpoonright r_0) \lrcorner && \text{for some } w\text{, by (4.6)} \\
&= w \cdot \llcorner t \upharpoonleft\!\upharpoonright r_0 \lrcorner && \text{by (4.4).}
\end{aligned}$$

Thus $\llcorner t \lrcorner \upharpoonleft\!\upharpoonright r_0 \sqsubseteq w \cdot \llcorner t \upharpoonleft\!\upharpoonright r_0 \lrcorner$. But by definition of the operator $\_ \upharpoonleft\!\upharpoonright$, both $\llcorner t \lrcorner \upharpoonleft\!\upharpoonright r_0$ and $\llcorner t \upharpoonleft\!\upharpoonright r_0 \lrcorner$ start with the occurrence r_0, we thus have $\llcorner t \lrcorner \upharpoonleft\!\upharpoonright r_0 \sqsubseteq \llcorner t \upharpoonleft\!\upharpoonright r_0 \lrcorner$. $\square$

Example 4.60. Take $\varphi : 2, e : o \vdash \varphi(\lambda x^o.(\lambda\psi^2.\varphi(\lambda x_1^o.(\lambda y^o.\psi(\lambda z^o.z))(\varphi(\lambda x_2^o.x_1))))(\lambda u^1.ue)) : o$. The computation tree is represented below together with an example of traversal t:



$$t = \lambda\ \varphi\ \lambda x\ @\ \lambda\psi\ \varphi\ \lambda x_1\ @\ \lambda y\ \psi\ \lambda u\ u\ \lambda z\ z\ \lambda$$

$$\llcorner t \lrcorner = @\ \lambda\psi\ \psi\ \lambda u\ u\ \lambda z\ z\ \lambda$$

$$\llcorner t \lrcorner \upharpoonright\!\upharpoonright \lambda\psi = \lambda\psi\ \psi\ \lambda z\ z$$

$$t \upharpoonright\!\upharpoonright \lambda\psi = \lambda\psi\ \varphi\ \lambda x_1\ @\ \lambda y\ \psi\ \lambda z\ z$$

$$\llcorner t \upharpoonright\!\upharpoonright \lambda\psi \lrcorner = \lambda\psi\ \psi\ \lambda z\ z\ .$$

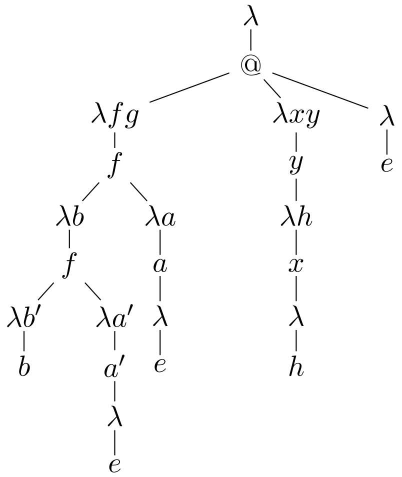
Example 4.61. Take the term-in-context:

$$e : o \vdash (\lambda f^{1\to 2\to o} g^o.f(\lambda b^o.f(\lambda c^o.b)(\lambda d^1.de))(\lambda a^1.ae))(\lambda x^1 y^2.y(\lambda h^o.xh))e : o\ \ .$$

Take the traversal:

$$t = \lambda\ @\ \lambda fg\ f\ \lambda xy\ y\ \lambda a\ a\ \lambda h\ x\ \lambda b\ f\ \lambda xy\ y\ \lambda d\ d\ \lambda h\ x\ \lambda c\ b\ \lambda\ h$$

and let r_0 denote the occurrence of the node λfg in t. We have the following relations:



$$\llcorner t \lrcorner = @\ \lambda fg\ f\ \lambda xy\ y\ \lambda a\ a\ \lambda h\ h$$

$$\llcorner t \lrcorner \upharpoonright\!\upharpoonright r_0 = \lambda fg\ f\ \lambda a\ a$$

$$t \upharpoonright\!\upharpoonright r_0 = \lambda fg\ f\ \lambda a\ a\ \lambda b\ f\ \lambda d\ d\ \lambda c\ b$$

$$\llcorner t \upharpoonright\!\upharpoonright r_0 \lrcorner = \lambda fg\ f\ \lambda a\ a\ \lambda b\ b\ .$$

4.1.3.8 Subterm projections are sub-traversals

The following proposition is a key step in relating the bottom-up presentation of game semantics with the top-down approach provided by the traversal theory. It states that the subterm projection of a traversal (see Sec. 4.1.3.6) is a traversal of the (computation tree of the) subterm. The proof relies on all the lemmas and propositions from the previous two sections.

Proposition 4.62 (Subterm projections are sub-traversals)**.** *Let $t \in \mathcal{T}rav(M)$. For every occurrence r_0 in t of some lambda node $r \in IN_\lambda$ we have $t \upharpoonright\!\upharpoonright r_0 \in \mathcal{T}rav(M^{(r)})$.*

Proof. We proceed by induction on the traversal rules. The base cases (Empty) and (Root) are trivial. *Step case:* Take a traversal $t \in \mathcal{T}rav(M)$ and suppose that the result holds for every traversal shorter than t. Suppose that t^ω does not appear in $t \upharpoonleft\!\upharpoonright r_0$ then the result follows by applying the induction hypothesis on the immediate prefix of t. Suppose that t^ω appears in $t \upharpoonleft\!\upharpoonright r_0$ then we do a case analysis on the last traversal rule used to form t:

- (Lam) We have $t = t' \cdot n$ with $t' = \ldots \cdot \lambda\overline{\xi}$. By the induction hypothesis, $t' \upharpoonleft\!\upharpoonright r_0 \in \mathcal{T}rav(M^{(r)})$. Since n is a variable node appearing in $t \upharpoonleft\!\upharpoonright r_0$, by definition of $t \upharpoonleft\!\upharpoonright r_0$ its immediate predecessor $\lambda\overline{\xi}$ must occur in $t \upharpoonleft\!\upharpoonright r_0$ and therefore must be the last occurrence in $t' \upharpoonleft\!\upharpoonright r_0$. Thus we can use the rule (Lam) in $\tau(M^{(r)})$ to produce the traversal $u = (t' \upharpoonleft\!\upharpoonright r_0) \cdot n$ of $M^{(r)}$.

We have $t \upharpoonleft\!\upharpoonright r_0 = (t' \upharpoonleft\!\upharpoonright r_0) \cdot n$, but in order to state the equality $u = t \upharpoonleft\!\upharpoonright r_0$ it remains to prove that n has the same link in $t \upharpoonleft\!\upharpoonright r_0$ and in u.
The case $n \in IN_@ \cup IN_\Sigma$ is trivial: n has no justifier in both u and $t \upharpoonleft\!\upharpoonright r_0$. Otherwise $n \in IN_{\mathsf{var}}$, let m_u denote the occurrence in t of n's justifier in u, m_t for the occurrence in t of n's justifier in t, and m for the occurrence in t of n's justifier in $t \upharpoonleft\!\upharpoonright r_0$. We want to show that $m_u = m$. By the rule (Var), m_u is defined as the only occurrence of n's enabler in $\ulcorner t' \upharpoonleft\!\upharpoonright r_0 \urcorner$ and m_t is the only occurrence of n's enabler in $\ulcorner t' \urcorner$. There are two cases: (i) If r_0 occurs before m_t then by Lemma 4.49(ii), m_t appears in $t \upharpoonleft\!\upharpoonright r_0$ thus by definition of $\_ \upharpoonleft\!\upharpoonright$ we have $m = m_t$. Moreover, since m_t appears in $t \upharpoonleft\!\upharpoonright r_0$, it must appear after r_0 by Lemma 4.49(i.a), thus since it is in the P-view at t', it must be in $\ulcorner t \urcorner_{\geqslant r_0}$ which is equal to $\ulcorner t' \upharpoonleft\!\upharpoonright r_0 \urcorner$ by Lemma 4.49(i.b). Hence we necessarily have $m_u = m_t$ (since r occurs only once in the P-view $\ulcorner t' \upharpoonleft\!\upharpoonright r_0 \urcorner$). (ii) If r_0 occurs after m_t then m_t does not appear in $t \upharpoonleft\!\upharpoonright r_0$ thus $m = r_0$ by definition of $\_ \upharpoonleft\!\upharpoonright$. Moreover by Lemma 4.49(i), n's binder occurs in the path from r to the root $\circledast$. Thus n is a free variable in $\tau(M^{(r)})$ and consequently the only enabler of n occurring in $\ulcorner t' \upharpoonleft\!\upharpoonright r_0 \urcorner$ is necessarily r_0: $m_u = r_0$.
This proves the equality $t \upharpoonleft\!\upharpoonright r_0 = u$ and thus $t \upharpoonleft\!\upharpoonright r_0$ is a valid traversal of $M^{(r)}$.

- (App) $t = \ldots \cdot \lambda\overline{\xi} \cdot @ \cdot n$. Since n appears in $t \upharpoonleft\!\upharpoonright r_0$, so does @ (by definition of $t \upharpoonleft\!\upharpoonright r_0$). Hence @ is the last occurrence in $t' \upharpoonleft\!\upharpoonright r_0$. By the induction hypothesis, $t' \upharpoonleft\!\upharpoonright r_0$ is a traversal of $\tau(M^{(r)})$ therefore we can use the rule (App) in $\tau(M^{(r)})$ to produce the traversal $(t' \upharpoonleft\!\upharpoonright r_0) \cdot n = t \upharpoonleft\!\upharpoonright r_0$ of $M^{(r)}$.

- (Value$^{@\mapsto\lambda}$) Take $t = t' \cdot \lambda\overline{\xi} \cdot @ \ldots v_@ \cdot v_{\lambda\overline{\xi}}$. The occurrence $v_{\lambda\overline{\xi}}$ appears $t \upharpoonleft\!\upharpoonright r_0$ therefore since r_0 is not a lambda node, its justifier $\lambda\overline{\xi}$ also appears in $t \upharpoonleft\!\upharpoonright r_0$. Moreover since @ and $v_@$ are hereditarily justified by $\lambda\overline{\xi}$, they must also appear in $t \upharpoonleft\!\upharpoonright r_0$.
By the induction hypothesis $t' \upharpoonleft\!\upharpoonright r_0$ is a traversal of $\tau(M^{(r)})$ therefore since the occurrence $\lambda\overline{\xi}$, @, $v_@$, $v_{\lambda\overline{\xi}}$ all appear in $t \upharpoonleft\!\upharpoonright r_0$ we can use the rule (Value$^{@\mapsto\lambda}$) in $M^{(r)}$ to form the traversal $(t' \upharpoonleft\!\upharpoonright r_0) \cdot n = t \upharpoonleft\!\upharpoonright r_0$ of $M^{(r)}$.

- (Value$^{\lambda\mapsto@}$) Take $t = t' \cdot @ \cdot \lambda\overline{z} \ldots v_{\lambda\overline{z}} \cdot v_@$. Again, since $v_@$ appears in $t \upharpoonleft\!\upharpoonright r_0$, necessarily the occurrences @, $\lambda\overline{z}$, $v_{\lambda\overline{z}}$ and $v_@$ must all appear in $t \upharpoonleft\!\upharpoonright r_0$. Hence using the I.H. and the rule (Value$^{\lambda\mapsto@}$) in $M^{(r)}$ we obtain that $t \upharpoonleft\!\upharpoonright r_0$ is a traversal of $M^{(r)}$.

- (Value$^{\mathsf{var}\mapsto\lambda}$) Take $t = t' \cdot \lambda\overline{\xi} \cdot x \ldots v_x \cdot v_{\lambda\overline{\xi}}$. Since $v_{\lambda\overline{\xi}}$ is in $t \upharpoonleft\!\upharpoonright r_0$, so must be x, v_x and $\lambda\overline{\xi}$, by definition of $t \upharpoonleft\!\upharpoonright r_0$. Hence we can use the I.H. to form the traversal $t \upharpoonleft\!\upharpoonright r_0$ of $M^{(r)}$.

- (InputValue) Take $t = t_1 \cdot x \cdot t_2 \cdot v_x$ for some $v \in \mathcal{D}$ where x is the pending node in $t_1 \cdot x \cdot t_2$ and $x \in IN_{\mathsf{var}}^{\circledast\vdash}$. Since v_x appears in $t \upharpoonleft\!\upharpoonright r_0$, so does x hence by Lemma 4.48, x is also the pending node in $(t_1 \cdot x \cdot t_2) \upharpoonleft\!\upharpoonright r_0$. Furthermore since $M^{(r)}$ is a subterm of M, x is necessarily an input-variable node in $\tau(M^{(r)})$. Hence we can conclude using the I.H. and the rule (InputValue).

- (InputVar) Take $t = t' \cdot n$ where $n \in IN_\lambda$ points to an occurrence of its parent node $y \in IN_{\mathsf{var}}^{\circledast\vdash}$ in $\llcorner t \lrcorner$. By Lemma 4.47(a), y must also appear in $t \upharpoonleft\!\upharpoonright r_0$, therefore y also occurs in $\llcorner t \upharpoonleft\!\upharpoonright r_0 \lrcorner \sqsubseteq \llcorner t \lrcorner \upharpoonleft\!\upharpoonright r_0$. Hence we can conclude using the rule (InputVar) in $M^{(r)}$.

- (Var) Take $t = t' \cdot p \cdot \lambda\overline{x} \ldots x_i \cdot \lambda\overline{\eta_i}$ for some variable x_i in $IN_{\mathsf{var}}^{@\vdash}$. If $\lambda\overline{\eta_i}$ is the occurrence r_0 then the traversal $t \upharpoonleft\!\upharpoonright r_0 = r_0$ can be formed using the rule (Root). Otherwise $\lambda\overline{\eta_i}$ is not the occurrence r_0. Then both $\lambda\overline{\eta_i}$ and its justifier p must appear in $t \upharpoonleft\!\upharpoonright r_0$. The nodes $\lambda\overline{x}$ and x_i, however, do not necessarily appear in $t \upharpoonright\!\upharpoonright r_0$. Consider the node @ that initiates the thread of $\lambda\overline{\eta_i}$.

– Suppose that r_0 precedes @ in t then by Lemma 4.52(i), the nodes $\lambda\overline{\eta_i}$, p, $\lambda\overline{x}$ and x_i as well as @ all appear in $t \upharpoonright\!\upharpoonright r_0$. Moreover since @ appear in $t \upharpoonright\!\upharpoonright r_0$, it must be an occurrence of an application node that appear in the subtree rooted at r thus $@ \in IN^{r\vdash}_{\mathsf{var}}$. Hence we can use the use the rule (Var) in $M^{(r)}$ to form the traversal $t \upharpoonright\!\upharpoonright r_0$ of $M^{(r)}$.
– Suppose that @ precedes r_0 in t then by Lemma 4.52(ii), p is necessarily an input variable node in $\tau(M^{(r)})$. We have $p \in \llcorner t \lrcorner \upharpoonright\!\upharpoonright r_0 \sqsubseteq \llcorner t \upharpoonright\!\upharpoonright r_0 \lrcorner$ by Proposition 4.53. Furthermore we can easily check (by alternation and using the fact that if an occurrence in $IN_\lambda \cup L_{\mathsf{var}} \cup L_@ \cup L_\Sigma \cup IN_@ \cup IN_\Sigma$ appears in $t \upharpoonright\!\upharpoonright r_0$ then so does its immediate successor) that the penultimate node in $t \upharpoonright\!\upharpoonright r_0$ is necessarily in $IN_{\mathsf{var}} \cup L_\lambda$. Hence we can make use of the rule (InputVar) in $M^{(r)}$ (in its alternative form) to produce the traversal $t \upharpoonright\!\upharpoonright r_0$ of $M^{(r)}$.

- (Value$^{\lambda\mapsto\mathsf{var}}$) Take $t = t' \cdot y \cdot \lambda\overline{\xi} \ldots v_{\lambda\overline{\xi}} \cdot v_y$ for some variable y in $IN^{@\vdash}_{\mathsf{var}}$. The proof is similar to the previous case using the rule (InputValue) instead of (InputVar) in the second subcase.
- $(\Sigma)/(\Sigma\text{-var})$ The proof is similar to the case (App) and (Var).
- (Σ-Value) The proof is similar to the case (Value$^{\lambda\mapsto\mathsf{var}}$). □

The following Lemma will be useful to prove the Correspondence Theorem:

Lemma 4.63. *Let t be a traversal and r_0 be an occurrence of a lambda node r. We have*

$$(t \upharpoonright\!\upharpoonright r_0)^\star = t^\star \upharpoonright N^{(r)} \upharpoonright r_0 \ .$$

Proof. By the previous Lemma, $t \upharpoonright\!\upharpoonright r_0$ is indeed a traversal (of $\tau(M^{(r)})$) thus the expression "$(t \upharpoonright\!\upharpoonright r_0)^\star$" is well-defined. We show the result by induction on t: It is true for the empty traversal. Take $t = t' \cdot n$.

If n does not belong to $N_@ \cup N_\Sigma$ then

$$((t' \cdot n) \upharpoonright\!\upharpoonright r_0)^\star = (t' \upharpoonright\!\upharpoonright r_0)^\star \cdot \begin{cases} n, & \text{if } n \text{ appears in } t \upharpoonright\!\upharpoonright r_0; \\ \epsilon, & \text{otherwise.} \end{cases}$$

and $((t' \cdot n)^\star \upharpoonright N^{(r)}) \upharpoonright r_0 = (t'^\star \upharpoonright N^{(r)}) \upharpoonright r_0 \cdot \begin{cases} n, & \text{if } n \text{ is her. just. by } r_0 \text{ in } t^\star \upharpoonright N^{(r)}; \\ \epsilon, & \text{otherwise.} \end{cases}$

Since $t^\omega \not\in N_@ \cup N_\Sigma$, by Lemma 4.51 we have that n is hereditarily justified by r_0 in $t^\star \upharpoonright N^{(r)}$ if and only if n appears in $t \upharpoonright\!\upharpoonright r_0$. Hence we can conclude using the I.H. on t'.

If n belongs to $N_@ \cup N_\Sigma$ then

$$\begin{aligned} ((t' \cdot n) \upharpoonright\!\upharpoonright r_0)^\star &= (t' \upharpoonright\!\upharpoonright r_0)^\star \\ &= (t'^\star \upharpoonright N^{(r)}) \upharpoonright r_0 && \text{by the I.H. on } t' \\ &= ((t' \cdot n)^\star \upharpoonright N^{(r)}) \upharpoonright r_0 && \square \end{aligned}$$

Together with Lemma 4.39 this implies that if $t^\omega \not\in N_@ \cup N_\Sigma$ then $t \upharpoonright\!\upharpoonright r_0 = (t^\star \upharpoonright r_0) + \Sigma + @$.

4.1.3.9 O-view and P-view projection with respect to the root

Lemma 4.64 (O-view projection with respect to the root)**.** *Let t be a non-empty traversal of M and r denote the only occurrence of $\tau(M)$'s root in t. If t^ω appears in $t \upharpoonright r$ then:*

$$\llcorner t \upharpoonright r \lrcorner = \llcorner t \lrcorner \upharpoonright r = \llcorner t \lrcorner \ .$$

Proof. It follows immediately from the fact that, by Lemma 4.32, all the occurrences in $\llcorner t \lrcorner$ belong to the same thread and therefore are all hereditarily justified by r. □

Lemma 4.65 (P-view projection with respect to the root)**.** *Let t be a non-empty traversal of M and r denote the only occurrence of $\tau(M)$'s root in t. If t^ω appears in $t \upharpoonright r$ then:*

$$\ulcorner t \urcorner \upharpoonright r \sqsubseteq \ulcorner t \upharpoonright r \urcorner \ .$$

Proof. We just sketch the proof. We proceed exactly in the same way as for the proof of Proposition 4.53. Again we establish an analogy between traversals and plays of game semantics:

Traversal setting	**Game-semantic setting**
Traversal t	Play s
O-nodes $(L_{\mathsf{var}} \cup L_\Sigma \cup L_@ \cup IN_\lambda)$	O-moves $\bullet$
P-Nodes $(IN_{\mathsf{var}} \cup IN_\Sigma \cup IN_@ \cup L_\lambda)$ or $\diamond$	P-moves $\circ$
P-view $\ulcorner t \urcorner$	P-view $\ulcorner s \urcorner$
O-view $\llcorner t \lrcorner$	O-view $\llcorner s \lrcorner$
Occurrence n her. just. by r in t	Occurrence $n \in B$
Occurrence n not her. just. by r in t	Occurrence $n \in A$
No notion of initiality (all nodes are considered to be non-initial).	Distinction between initial and non-initial move.

Clearly the conditions (w1) to (w8) hold. Hence we can reuse Proposition 4.3 form [HO00] which gives the desired result. ☐

The previous result only gives us an inequality. In the particular case where interpreted constants are well-behaved, however, and if we consider the subsequence of a traversal consisting of unanswered nodes only, then we obtain an equality:

Lemma 4.66. *Suppose that M is in β-normal form and all the Σ-constants are well-behaved. Let t be a non-empty traversal of M and r denote the only occurrence in t of $\tau(M)$'s root.*

(a) If t's last occurrence is not a leaf then $\ulcorner t \urcorner \upharpoonright r = \ulcorner ?(t) \upharpoonright r \urcorner = \ulcorner ?(t \upharpoonright r) \urcorner = ?(\ulcorner t \upharpoonright r \urcorner)$;

(b) If t's last occurrence is not a leaf and is hereditarily justified by r then $\ulcorner t \urcorner \upharpoonright r = \ulcorner t \upharpoonright r \urcorner$.

Proof. (a) It is easy to show that $?(t) \upharpoonright r = ?(t \upharpoonright r)$. This implies the second equality. The third equality can be shown by an easy induction and by observing that in a traversal core, variable occurrences are always immediately preceded by a lambda node (and not by a leaf). We show the first equality by induction. The base case $t = \epsilon$ is trivial. Consider a traversal t and suppose that the property is satisfied for all traversals shorter than t. Observe that since t contains at most a single occurrence r of the root $\circledast$, an occurrence n in t is hereditarily justified by r if and only if the corresponding node in $\tau(M)$ is hereditarily enabled by $\circledast$. Thus $t \upharpoonright r = t \upharpoonright IN^{\circledast\vdash}$. We do a case analysis on t's last node:

- $t^\omega \in IN_@$. This case does not happen since M is β-normal.
- $t = t' \cdot n$ with $n \in IN_{\mathsf{var}} \cup IN_\Sigma$ then t'^ω is not a leaf (otherwise n would also be a leaf by rule (Value)) thus we can use the I.H. on t' which, by an easy calculation, gives the desired equality.

Suppose that t^ω is a lambda node. There are three subcases:

- $t^\omega \in IN_\lambda^{@\vdash}$. Since the term is in β-normal form, there is no @-node in $\tau(M)$ so the rules (App) and (Var) are unused, hence this case does not happen.
- $t^\omega \in IN_\lambda^{IN_\Sigma\vdash}$. We have $t = t' \cdot m \cdot u \cdot n$ with $n \in IN_\lambda^{IN_\Sigma\vdash}$ and $m \in IN_{\mathsf{var}} \cup IN_\Sigma$. The occurrence n is necessarily visited with a (Σ)-rule. Since, by assumption, these rules are well-behaved we have $?(u) = \epsilon$. Hence:

$$\ulcorner t \urcorner \upharpoonright r = \ulcorner t' \cdot m \cdot u \cdot n \urcorner \upharpoonright r \qquad \text{(def. of } t\text{)}$$

$$
\begin{aligned}
&= (\ulcorner t' \urcorner \cdot m \cdot n) \upharpoonright r && \text{(P-view computation)}\\
&= \ulcorner t' \urcorner \upharpoonright r && (m, n \notin IN^{\circledast\vdash})\\
&= \ulcorner ?(t') \upharpoonright r \urcorner && \text{(induction hypothesis)}\\
&= \ulcorner ?(t' \cdot m \cdot n) \upharpoonright r \urcorner && (m, n \notin IN^{\circledast\vdash})\\
&= \ulcorner ?(t' \cdot m \cdot u \cdot n) \upharpoonright r \urcorner && (?(u) = \epsilon)\\
&= \ulcorner ?(t) \upharpoonright r \urcorner && (\text{since } u = \epsilon).
\end{aligned}
$$

- $t^\omega \in IN_\lambda^{\circledast\vdash}$. If $t = r$ then the result holds trivially. Otherwise $t = t' \cdot m \cdot u \cdot n$ for some $n \in IN_\lambda^{\circledast\vdash}$. An easy calculation using the induction hypothesis on $t' \cdot m$ shows the desired equality.

(b) If t's last occurrence is hereditarily justified by r then the last occurrence of $t \upharpoonright r$ is precisely the last occurrence of t and is therefore not a leaf. In a traversal core, variable nodes are immediately preceded by lambda nodes thus since the last node in $t \upharpoonright r$ is not a leaf, an easy induction shows that all the nodes in $\ulcorner t \upharpoonright r \urcorner$ are not leaves. Consequently $?(\ulcorner t \upharpoonright r \urcorner) = \ulcorner t \upharpoonright r \urcorner$. □

The hypothesis that the term is beta-normal is crucial in this Lemma. Take for instance the term $\lambda x^o f^{(o,o)}.(\lambda y^o.f\, y)x$. A possible traversal is

$$t = \lambda xf \cdot @ \cdot \lambda y \cdot f \cdot \lambda \cdot y \cdot \lambda \cdot x \ .$$

But $\ulcorner t \urcorner \upharpoonright r = \lambda xf \cdot x$ is only a strict subsequence of $\ulcorner t \upharpoonright r \urcorner = \lambda xf \cdot f \cdot \lambda \cdot x$.

4.2 Game semantics correspondence

We work in the general setting of an applied simply-typed lambda calculus with a given set of higher-order constants Σ. The operational semantics of these constants is given by certain reduction rules. We assume that a fully abstract model of the calculus is provided by means of a category of well-bracketed games. For instance, if Σ consists of the PCF constants then we work in the category of games and innocent well-bracketed strategies [HO00, AMJ94]. A strategy is commonly defined in the literature as a set of plays closed by even-length prefixing. For our purpose, however, it is more convenient to represent strategies using *prefix-closed* set of plays. This will spare us some considerations on the parity of traversal length when showing the correspondence between traversals and game semantics. For the rest of the section we fix a simply-typed term $\Gamma \vdash M : T$. We write $[\![\Gamma \vdash M : T]\!]$ for its strategy denotation (in the standard cartesian closed category of games and innocent strategies [AMJ94, HO00]). We use the notation $\mathsf{Pref}(S)$ to denote the prefix-closure of the set S.

4.2.1 Revealed game semantics

In standard game semantics, terms are denoted by strategies that are computed inductively on the structure of the term: calculating the denotation of a term boils down to performing the composition of strategies denoting some of its subterms. Strategy composition is the CSP-like "composition + hiding" operation where all the internal moves are hidden.

It is possible to use an alternative notion of composition where the internal moves are not hidden. Game model based on such notion of composition have appeared in the literature under the name *revealed semantics* [Gre04] and *interaction semantics* [DGL05]. In such game models, the denotation is computed inductively on the syntax of the term as in the standard game semantics, but certain internal moves may be uncovered after composition. There is not just one revealed semantics as one may desire to hide/uncover different internal moves. Such semantics will help to establish a correspondence between the game semantics of a term and the traversals of its computation tree.

This section presents a general setting in which revealed semantics can be defined. At the end of the section we will provide an example of a revealed semantics that is calculated inductively on the syntax of the η-long normal form of the term.

4.2.1.1 Revealed strategies

Definition 4.67. We consider ordered trees whose leaves are labelled with PCF simple types and inner nodes are labelled with symbols in $\{;, \langle \_ , \_ \rangle, \Lambda\}$ where '$;$' and '$\langle \_ , \_ \rangle$' are of arity 2 and 'Λ' is of arity one. We write $\langle T_1, T_2\rangle$ for the tree obtained by attaching T_1 and T_2 to a $\langle \_ , \_ \rangle$-node, and similarly we use the notations $T_1;T_2$ and $\Lambda(T_1)$.

The set of ***interaction type trees***, or just ***interaction types***, is defined inductively as follows:

- *Leaf*: If T is a leaf annotated by a type A then T is an interaction type, and we define $type(T)$ to be A;
- *Currying*: If T is an interaction type with $type(T) = A \times B \rightarrow C$ then $\Lambda(T)$ is also an interaction type and $type(\Lambda(T)) = A \rightarrow (B \rightarrow C)$;
- *Pairing*: If T_1 and T_2 are interaction types with $type(T_1) = C \rightarrow A$ and $type(T_2) = C \rightarrow B$ then $\langle T_1, T_2\rangle$ is also an interaction type and $type(\langle T_1, T_2\rangle) = C \rightarrow A \times B$. Pairing generalizes straightforwardly to a p-tuple operator $\langle \Sigma_1, \ldots, \Sigma_p\rangle$ for $p \geq 2$, in which case the tree has p child subtrees;
- *Composition*: If T_1 and T_2 are interaction types with $type(T_1) = A \rightarrow B$ and $type(T_2) = B \rightarrow C$ then $T_1;T_2$ is also an interaction type and $type(T_1;T_2) = A \rightarrow C$.

We call $type(T)$ the ***underlying type*** (or just type) of the interaction type T. We sometimes write T^A to indicate that $type(T) = A$.

Let T be an interaction type tree. Each node of type A in T can be mapped to the (standard) game $[\![A]\!]$. By taking the image of T across this mapping we obtain a tree whose leaves and nodes are labelled by games. This tree, written $\langle\!\langle T\rangle\!\rangle$, is called an ***interaction game***. A ***revealed strategy*** Σ on the interaction game $\langle\!\langle T\rangle\!\rangle$ is a composition of several standard strategies in which certain internal moves are not hidden. Formally:

Definition 4.68. A ***revealed strategy*** Σ on an interaction game $\langle\!\langle T\rangle\!\rangle$, written $\Sigma : \langle\!\langle T\rangle\!\rangle$, is an annotated interaction type tree T where

- each leaf $[\![A]\!]$ of T is annotated with a (standard) strategy σ on the game $[\![A]\!]$;
- each ;-node is annotated with two sets of indices $S, P \subseteq \mathbb{N}$ called respectively the *superficial* and *profound* uncovering indices.

The intuition behind this definition is that if a ;-node has children $\Sigma_1 : \langle\!\langle A \rightarrow B\rangle\!\rangle$ and $\Sigma_2 : \langle\!\langle B \rightarrow C\rangle\!\rangle$ then the two sets of indices S, P indicate which components of B should be uncovered when performing composition. The set S indicates which ***superficial*** internal moves (*i.e.*, those that are created by the top-level composition between Σ_1 and Σ_2) to uncover; whereas the set P indicates the ***profound*** internal moves (*i.e.*, those that are already present in the revealed strategies Σ_1 and Σ_2) to uncover. This notion of uncovering is made concrete in the next paragraph where we define *revealed strategies* by means of *uncovered positions*.

Example 4.69. The diagrams below represent an interaction type tree T (left), the corresponding interaction game $\langle\!\langle T\rangle\!\rangle$ (middle) and a revealed strategy Σ (right):

$$
\begin{array}{ccccc}
 & & ; & & \\
 & / & & \backslash & \\
 & ; & & & C \to D \\
 / & & \backslash & & \\
A \to B & & B \to C & &
\end{array}
\qquad
\begin{array}{ccccc}
 & & ; & & \\
 & / & & \backslash & \\
 & ; & & & [\![C \to D]\!] \\
 / & & \backslash & & \\
[\![A \to B]\!] & & [\![B \to C]\!] & &
\end{array}
\qquad
\begin{array}{ccccc}
 & & ;^{\{0\},\{0\}} & & \\
 & / & & \backslash & \\
 & ;^{\emptyset,\{0\}} & & & (C \to D)^{\sigma_3} \\
 / & & \backslash & & \\
(A \to B)^{\sigma_1} & & (B \to C)^{\sigma_2} & &
\end{array}
$$

For convenience, a revealed strategy will be written as an expression in infix form: for instance the strategy of the example above is written $\Sigma = (\sigma_1;^{\emptyset,\{0\}} \sigma_2);^{\{0\},\{0\}} \sigma_3$.

A revealed strategy induces a strategy in the usual sense: the standard strategy $\sigma : A$ ***induced*** by a reveled strategy $\Sigma : T^A$ is obtained by replacing each occurrence of the operator '$;^{S,P}$' for some S, P by '$;^{\emptyset,\emptyset}$' (also abbreviated '$;$') in the expression of Σ. For instance the strategy Σ from the example above induces the strategy $(\sigma_1; \sigma_2); \sigma_3$ on the game $A \to D$.

4.2.1.2 Uncovered play

The analogue of a play in the revealed semantics is called an *uncovered play* or *uncovered position*; it is a play whose moves are interleaved with internal moves. Each move in such a play may belong to multiple games from different nodes of the interaction game; they are thus implicitly tagged so that one can retrieve the components of the node-games to which the move belongs.

Definition 4.70. The ***set of possible moves*** M_T of an interaction game $\langle\!\langle T \rangle\!\rangle$ is defined as $\mathcal{M}_T/{\sim_T}$, the quotient of the set $\mathcal{M}_T$ by the equivalence relation $\sim_T \subseteq \mathcal{M}_T \times \mathcal{M}_T$ defined as follows: For a single leaf tree T labelled by a type A we define $\mathcal{M}_T = M_A$ and $\sim_T = id_{M_A}$; for other cases:

$$
\begin{aligned}
\mathcal{M}_{\Lambda(T^{A \times B \to C})} &= \mathcal{M}_T + M_{A \to B \to C} \\
\sim_{\Lambda(T^{A \times B \to C})} &= (\sim_T \cup ((A \times B \to C) \leftrightarrow (A \to (B \to C))))^{=}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{M}_{\langle T_1^{C^1 \to A^1}, T_2^{C^2 \to B^2} \rangle} &= \mathcal{M}_{T_1} + \mathcal{M}_{T_2} + M_{C \to (A \times B)} \\
\sim_{\langle T_1^{C^1 \to A^1}, T_2^{C^2 \to B^2} \rangle} &= \big(\sim_{T_1} \cup \sim_{T_2} \cup (C^1 \leftrightarrow C) \cup (C^2 \leftrightarrow C) \cup (A^1 \leftrightarrow A) \cup (B^2 \leftrightarrow B)\big)^{=}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{M}_{T_1^{A \to B}; T_2^{B \to C}} &= \mathcal{M}_{T_1} + \mathcal{M}_{T_2} + M_{A \to C} \\
\sim_{T_1^{A^1 \to B^1}; T_2^{B^2 \to C^2}} &= \big(\sim_{T_1} \cup \sim_{T_2} \cup (A^1 \leftrightarrow A) \cup (B^1 \leftrightarrow B^2) \cup (C \leftrightarrow C^2)\big)^{=}
\end{aligned}
$$

where $A \leftrightarrow B$ denotes the canonical bijection between M_A and M_B for two isomorphic games A and B; and $R^{=}$ denotes the smallest equivalence relation containing R.

It is easy to check that for every sub-type tree T' of T, the equivalence classes of $M_{T'}$ are subsets of equivalence classes of M_T. Thus $M_{T'}$ can be viewed as a subset of M_T.

We call ***internal move*** of the game $\langle\!\langle T \rangle\!\rangle$, any $\sim$-class from M_T that does not contain any move from $M_{type(T)}$. We denote the set of all internal moves by M_T^{int}. The complement of M_T^{int} in M_T, called the set of ***external moves***, is denoted by M_T^{ext}. For every subgame A occurring in some node of the interaction game T, we write $M_{T,A}^{\mathsf{int}}$ (resp. $M_{T,A}^{\mathsf{ext}}$) for the subset of moves of M_T^{int} (resp. M_T^{ext}) consisting of $\sim$-classes containing some move in M_A.

A ***justified interaction sequence*** of moves on the interaction game $\langle\!\langle T \rangle\!\rangle$ is a sequence of moves from M_T together with pointers where each move in the sequence except the first one has a link attached to it pointing to some preceding move in the sequence. We write J_T to denote the set of justified interaction sequences over $\langle\!\langle T \rangle\!\rangle$.

Definition 4.71 (Projection)**.** Let $s \in J_T$ for some interaction game T. We define the following projection operations:

(a) Let M' be a subset of M_T. The projection $s \upharpoonright M'$ is defined as the subsequence of s consisting of $\sim$-equivalence classes from M';

(b) Let A be a sub-game of $[\![type(T)]\!]$. We define the projection operator $s \upharpoonright A$ to be the subsequence of s consisting of the $\sim$-classes that contain some move in M_A. Formally $s \upharpoonright A \stackrel{\text{def}}{=} s \upharpoonright \{[m] \mid m \in M_A\}$ where $[m]$ denotes the $\sim$-equivalence class of m.

(c) Let m be a $[\![type(T)]\!]$-initial move occurring in s. We define $s \upharpoonright m$ as the subsequence of s consisting of moves that are *hereditarily justified* by that occurrence of m in $s \upharpoonright [\![type(T)]\!]$.

(d) Let T' be an immediate subtree of T. The projection $s \upharpoonright T'$ is defined as follows:

   (i) the sequence $s \upharpoonright T'$ viewed as a sequence of moves without pointers is defined as $s \upharpoonright M_{T'}$ (*i.e.*, the subsequence of s consisting of the $\sim$-equivalence classes that contain some equivalence class of $M_{T'}$; see (a));

   (ii) the justification pointers of $s \upharpoonright T'$ are those of s except that if an element m loses its pointer (*i.e.*, if its justifier does not appear in $s \upharpoonright T'$) then its justifier is redefined as the only occurrence of an initial $[\![type(T')]\!]$-move in $\ulcorner s \upharpoonright M_{T'} \upharpoonright [\![type(T')]\!] \urcorner$ (*cf.* (a) and (b)).

(e) Let T' be a non-immediate subtree of T. We define the projection $s \upharpoonright T'$ as $(\ldots(s \upharpoonright T^0) \upharpoonright \ldots \upharpoonright T^{k-1}) \upharpoonright T^k$ where $T^0, \ldots, T^k$ is the uniquely defined sequence of subtrees of T satisfying $T = T^0$, $T' = T^k$ and such that for every $1 \leq l \leq k$, T^l is an immediate subtree of T^{l-1}.

(f) Let T' be some subtree of T and A be a sub-game of $[\![type(T')]\!]$. Then we write $s \upharpoonright A$ for $s \upharpoonright T' \upharpoonright A$.

By extension, we also define these operations on *sets* of justified interaction sequences.

We now characterize revealed strategies by means of sets of justified sequences of moves called *uncovered positions* or *uncovered plays*. This set is calculated by a bottom-up computation on the strategy tree. At each ;-node, we apply the composition operation of game semantics. In accordance with standard game semantics, justification pointers are adjusted when composing two interaction strategies $\Sigma_l : T_l^{A \to B}$ and $\Sigma_r : T_r^{B \to C}$: if an initial A-move a is justified by an initial B-move itself justified by an initial C-move c then a's justifier is set to c (see definition of the projection $\_ \upharpoonright A, C$ [AM98b]). This guarantees that for every interaction position u of $\Sigma_l ; \Sigma_r$, the subsequence consisting of moves in A and C only—filtering out B-moves as well as the internal moves coming from compositions taking place at deeper level in the revealed semantics—is a valid position of the *standard* strategy underlying $\Sigma_l ; \Sigma_r$. In contrast with the standard game semantics, however, not all internal moves are hidden during composition.

Definition 4.72. A revealed strategy Σ (defined by means of an annotated type tree) is characterized by its set of ***uncovered positions*** defined inductively as follows:

- *Leaf* labelled with type A and annotated by the strategy σ: The set of positions of the revealed strategy is precisely the set of positions of the standard strategy σ.

- *Currying*: Let $\Sigma : \langle\!\langle T \rangle\!\rangle$.
$$\Lambda(\Sigma) = \{u \in J_{\Lambda(T)} \mid \rho(u) \in \Sigma\} \ ,$$
where ρ denotes the canonical bijection from $M_{\Lambda(T)}$ to M_T.

- *Pairing*: Let $\Sigma_1 : \langle\!\langle T_1 \rangle\!\rangle$ and $\Sigma_2 : \langle\!\langle T_2 \rangle\!\rangle$.
$$\begin{aligned} \langle \Sigma_1, \Sigma_2 \rangle \ = \ \{u \in J_{\langle T_1, T_2 \rangle} \mid \ & (u \upharpoonright T_1 \in \Sigma_1 \wedge \ u \upharpoonright T_2 = \epsilon) \\ & \vee \ (u \upharpoonright T_1 = \epsilon \wedge u \upharpoonright T_2 \in \Sigma_2)\} \ . \end{aligned}$$

- *Uncovered composition*: Let $\Sigma_1 : \langle\!\langle T_1 \rangle\!\rangle$ and $\Sigma_2 : \langle\!\langle T_2 \rangle\!\rangle$ where $type(T_1) = A \to B_0 \times \ldots \times B_l$ and $type(T_2) = B_0 \times \ldots \times B_l \to C$.

$$\begin{aligned}\Sigma_1 \| \Sigma_2 = \{u \in J_{T_1;T_2} \mid\ & u \upharpoonright T_2 \in \Sigma_2 \\ & \wedge \text{ for all occurrence } b \text{ in } u \text{ of an initial } [\![type(T_1)]\!]\text{-} \\ & \quad \text{move, } u \upharpoonright T_1 \upharpoonright b \in \Sigma_1 \\ & \wedge \text{ for every initial } A\text{-move } a \text{ justified in } u \upharpoonright T_1 \text{ by} \\ & \quad b \in B_j, \text{ itself justified by } c \in C \text{ in } u \upharpoonright T_2, \text{ we have} \\ & \quad \text{that } m \text{ is justified by } c \text{ in } u.\ \}\quad .\end{aligned}$$

- *Partially covered composition*: Let $\Sigma_1 : \langle\!\langle T_1 \rangle\!\rangle$ and $\Sigma_2 : \langle\!\langle T_2 \rangle\!\rangle$ where $type(T_1) = A \to B_0 \times \ldots \times B_l$ and $type(T_2) = B_0 \times \ldots \times B_l \to C$.

$$\begin{aligned}\Sigma_1 \,;^{S,P}\, \Sigma_2 &= \{\mathsf{hide}(u, \{0..l\} \setminus S, \{0..l\} \setminus P) \mid u \in \Sigma_1 \| \Sigma_2\} \\ \text{where } \mathsf{hide}(u,S,P) &= u \upharpoonright (M_T \setminus H(S,P)) \\ H(S,P) &= \bigcup_{j \in S} \underbrace{M^{\mathsf{ext}}_{T_1,B_j} \cup M^{\mathsf{ext}}_{T_2,B_j}}_{\text{superficial } B_j\text{-moves}} \cup \bigcup_{j \in P} \underbrace{M^{\mathsf{int}}_{T_1,B_j} \cup M^{\mathsf{int}}_{T_2,B_j}}_{\text{profound } B_j\text{-moves}} \quad .\end{aligned}$$

Observe that in particular $\Sigma_1 \| \Sigma_2 = \Sigma_1 ;^{\{0..l\},\{0..l\}} \Sigma_2$.

In words, the *uncovered composition* of $\Sigma_1 \|\ \Sigma_2$ is the set of uncovered plays obtained by performing the usual composition of the standard strategies underlying Σ_1 and Σ_2 while preserving the internal moves already in Σ_1 and Σ_2 as well as the internal moves produced by the composition itself.

On the other hand, given a product game $B = B_0 \times \ldots \times B_l$, the *partially covered composition* $\Sigma_1 ;^{S,P} \Sigma_2$ keeps only the superficial internal moves from the component B_k for $k \in S$ as well as the profound internal moves from the component B_k for $k \in P$.

As expected, this notion of set of uncovered positions is coherent with the usual notion of positions of a standard strategy:

Lemma 4.73. *Let $\Sigma : T$ be a revealed strategy* inducing *the standard strategy $\sigma : [\![type(T)]\!]$. Then for all $u \in \Sigma$, $u \upharpoonright [\![type(T)]\!] \in \sigma$.*

Proof. The proof is by induction on the structure of Σ. It follows from the fact that the operations on revealed strategies from Def. 4.72 are defined identically to their counterparts in the standard game semantics. ☐

4.2.1.3 Fully-revealed and syntactically-revealed semantics

We call *revealed semantics* any game model of a language in which a term is denoted by some revealed strategy as defined in the previous section. As we have already observed, depending on the internal moves that we wish to hide, we obtain different possible revealed strategies for a given term. Thus there is not a unique way to define a revealed semantics. In this section we give two examples of such semantics.

Let π_i denote the i^{th} projection strategy $\pi_i : [\![X_1 \times \ldots \times X_l]\!] \to [\![X_i]\!]$.

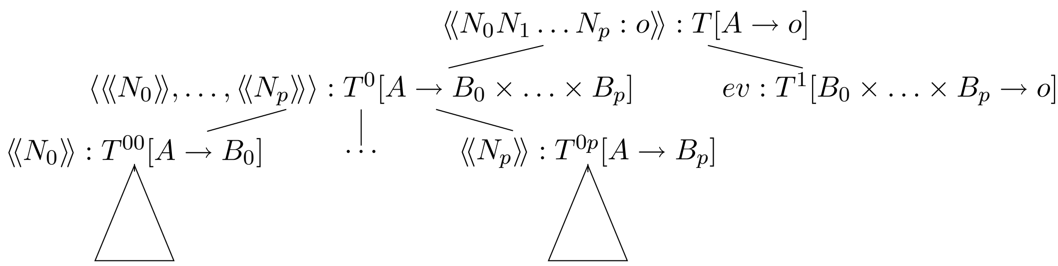
Definition 4.74 (The fully-revealed semantics)**.** The ***fully-revealed game denotation*** of M written $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle$ is defined by structural induction on the *η-long normal form of M*:

$$\begin{aligned}\langle\!\langle \Gamma \vdash \alpha : o \rangle\!\rangle &= [\![\Gamma \vdash \alpha : o]\!] \quad \text{where } \alpha \in \Gamma \cup \Sigma, \\ \langle\!\langle \Gamma \vdash \lambda\overline{\xi}.M : A \rangle\!\rangle &= \Lambda^{|\overline{\xi}|}(\langle\!\langle \Gamma, \overline{\xi} \vdash M : o \rangle\!\rangle) \\ \langle\!\langle \Gamma \vdash x_i N_1 \ldots N_p : o \rangle\!\rangle &= \langle \pi_i, \langle\!\langle \Gamma \vdash N_1 : A_1 \rangle\!\rangle, \ldots, \langle\!\langle \Gamma \vdash N_p : A_p \rangle\!\rangle \rangle \| ev^p, \quad X_i = A_0\end{aligned}$$

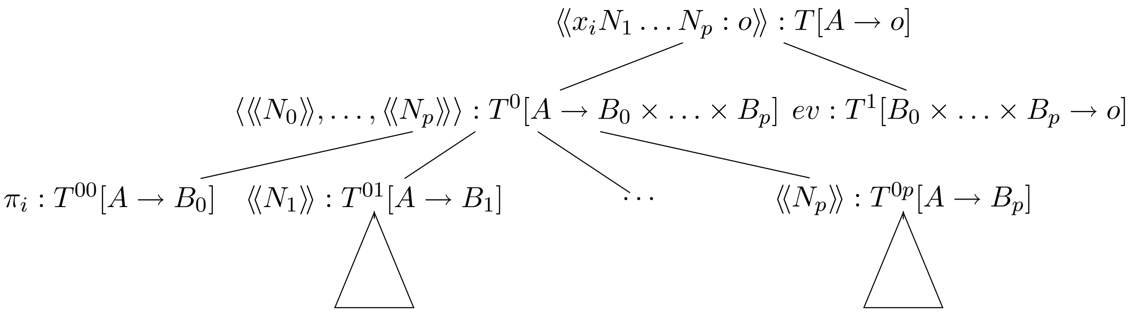
$$
\begin{array}{rcll}
\langle\!\langle \Gamma \vdash f N_1 \ldots N_p : o \rangle\!\rangle & = & \langle \langle\!\langle \Gamma \vdash N_1 : A_1 \rangle\!\rangle, \ldots, \langle\!\langle \Gamma \vdash N_p : A_p \rangle\!\rangle \rangle \parallel [\![f]\!], & f : A_0 \in \Sigma \\
\langle\!\langle \Gamma \vdash N_0 \ldots N_p : o \rangle\!\rangle & = & \langle \langle\!\langle \Gamma \vdash N_0 : A_0 \rangle\!\rangle, \ldots, \langle\!\langle \Gamma \vdash N_p : A_p \rangle\!\rangle \rangle \parallel ev^p &
\end{array}
$$

where $\Gamma = x_1 : X_1 \ldots x_l : X_l$, $A_0 = (A_1, \ldots, A_p, o)$ and ev^p denotes the evaluation strategy with p parameters where $p \geq 1$.

Fig. 4.1 shows tree representations of the interaction games involved in the revealed strategy $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle$ for the two application cases. These trees give us information about the constituent strategies involved in $\langle\!\langle M \rangle\!\rangle$. For instance the revealed strategy $\langle\!\langle N_0 \rangle\!\rangle$ is defined on the interaction game $\langle\!\langle T^{00} \rangle\!\rangle$ whose root game is $A \to B_0$, and the strategy ev is defined on the interaction game $\langle\!\langle T^1 \rangle\!\rangle$ whose underlying tree is constituted of a single game-node $B_0 \times \ldots \times B_p \to o$.



Tree-representation of the revealed strategy $\langle\!\langle \Gamma \vdash N_0 N_1 \ldots N_p : o \rangle\!\rangle$.



Tree-representation of the revealed strategy $\langle\!\langle \overline{x} : \overline{X} \vdash x_i N_1 \ldots N_p : o \rangle\!\rangle$.

A node label '$\Pi : T[G]$' indicates that Π is a revealed strategy on the interaction game T whose top-level game (at the root of the tree underlying T) is G. Each game is annotated with a string $s \in \{0..p\}^*$ in the exponent to indicate the path from the root to the corresponding node in the tree. (The digits in s tell the direction to take at each branch of the tree.)
The games A and B are given by:

$$
\begin{array}{rcl}
A & = & X_1 \times \ldots \times X_n \\
B & = & \underbrace{((B'_1 \times \ldots \times B'_p) \to o')}_{B_0} \times B_1 \times \ldots \times B_p \ .
\end{array}
$$

Figure 4.1: Tree-representation of the revealed strategy in the application case.

Example 4.75. Take the term $\lambda x^o.(\lambda f^{o \to o}.f x)(\lambda y^o.y)$. Its fully-revealed denotation is

$$\Lambda(\langle [\![x : X \vdash \lambda f^{o\to o}.f x : (o \to o) \to o]\!], [\![x : X \vdash \lambda y^o.y : o \to o]\!] \rangle \| ev^2) \ .$$

Note that the set of fully-revealed strategies does not give rise to a category because strategy composition is not associative and there is no identity interaction strategy.

Definition 4.76 (Syntactically-revealed semantics)**.** The ***syntactically-revealed game denotation*** of M written $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle_{\mathsf{s}}$ is defined by structural induction on the *η-long normal form of M*. The equations are the same as in Def. 4.74 except for the third case:

$$\langle\!\langle \Gamma \vdash x_i N_1 \ldots N_p : o \rangle\!\rangle_{\mathsf{s}} \quad = \quad \langle \pi_i, \langle\!\langle \Gamma \vdash N_1 : A_1 \rangle\!\rangle_{\mathsf{s}}, \ldots, \langle\!\langle \Gamma \vdash N_p : A_p \rangle\!\rangle_{\mathsf{s}} \rangle ;^{\emptyset,\{1..p\}} ev^p, \quad X_i = A_0 \ .$$

The syntactically-revealed denotation differs from the fully-revealed one in that only certain internal moves are preserved during composition: when computing the denotation of an application (joint by an @-node) in the computation tree, all the internal moves are preserved. However when computing the denotation of $\langle\!\langle x_i N_1 \ldots N_p \rangle\!\rangle_{\mathsf{s}}$ for some variable x_i, we only preserve the internal moves of $N_1, \ldots, N_p$ while omitting the internal moves produced by the copy-cat projection strategy denoting x_i.

4.2.1.4 Relating the two revealed denotations

As one would expect, the two revealed denotations that we have just introduced are in fact equivalent. We now show how $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle$ can be obtained from $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle_{\mathsf{s}}$ and conversely.

Fully-uncovered composition versus partially-uncovered composition In this paragraph we relate the fully-uncovered composition '$\|$' with the partially-uncovered composition '$;^{\emptyset,\{1..p\}}$' used in the definition of the syntactically-revealed semantics. Take a term $M \equiv x_i N_1 \ldots N_p$. Its revealed denotation is given by $\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}} = \Sigma_s ;^{\emptyset,\{1..p\}} ev$ where $\Sigma_s = \langle \pi_i, \langle\!\langle \Gamma \vdash N_1 : B_1 \rangle\!\rangle_{\mathsf{s}}, \ldots, \langle\!\langle \Gamma \vdash N_p : B_p \rangle\!\rangle_{\mathsf{s}} \rangle$. We use the notations introduced in Fig. 4.1: the composition takes place on the game

$$X_1 \times \ldots \overbrace{((B''_1 \times \ldots \times B''_p) \rightarrow o'')}^{X_i} \ldots \times X_n \xrightarrow{\Sigma} \boxed{\overbrace{((B'_1 \times \ldots \times B'_p) \rightarrow o')}^{B_0} \times B_1 \times \ldots \times B_p} \xrightarrow{ev} o$$

where the dashed-line frame contains the internal components of the game.

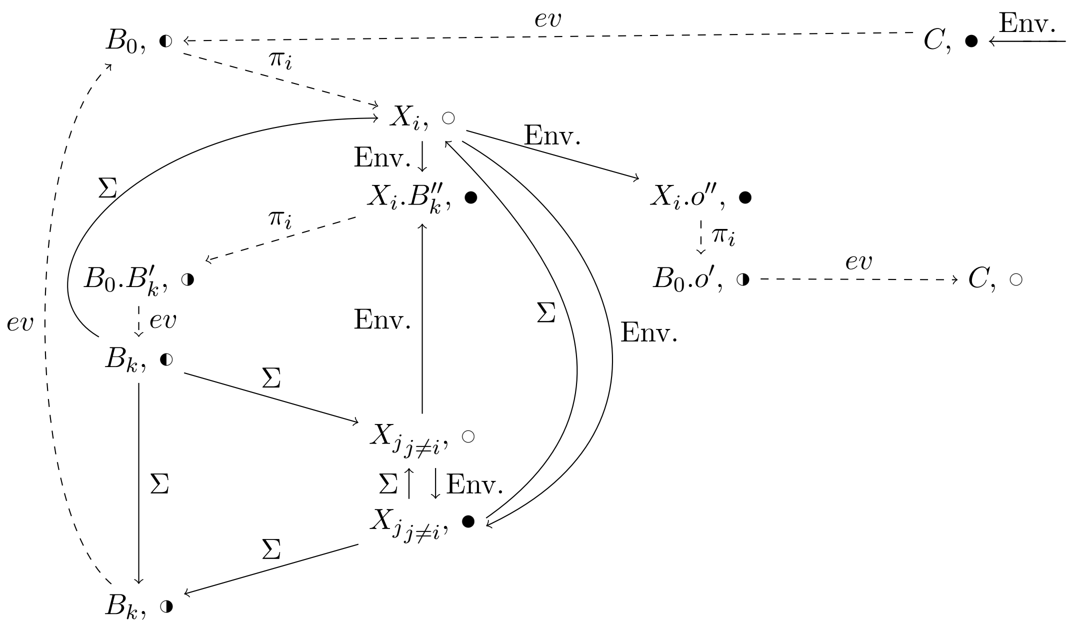
In $\Sigma_s \| ev$, all the internal moves from B_k for $k \in \{0..p\}$ are preserved, whereas in $\langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$, the internal B_0-moves as well as the superficial internal B_k-moves for $k \in \{1..p\}$ are hidden. By definition of the composition operator '$;^{\emptyset,\{1..p\}}$', the set $\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}}$ is obtained from $\Sigma_s \| ev$ by eliminating the internal B-moves appropriately:

$$\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}} = \Sigma_s ;^{\emptyset,\{1..p\}} ev = \{\mathsf{hide}(u, \{1..p\}, \{0\}) \mid u \in \Sigma_s \| ev\} \ .$$

We now show that conversely, there exists a transformation mapping the set $\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}}$ to $\Sigma_s \| ev$. More precisely we show that for every $u \in \langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}}$, there is a unique play v of $\Sigma_s \| ev$ ending with an external move such that eliminating the superficial internal moves from it gives us back u.

Let us look at the structure of an interaction play of $\Sigma \| ev$. The state-diagram in Fig. 4.2 describes precisely the flow of an interaction play. A node of the diagram indicates the last move that was played. Its label is of the form 'A, α' where A is the game in which the move was played, and $\alpha \in \{ \bullet, \circ, ◑, ◐ \}$ specifies the player that made the move. We use the symbols ◐, ◑, $\bullet$, $\circ$ for OP-move, PO-move, O-move and P-move respectively. We use the notation '$X_i.B''_k$' to denote the sub-component B''_k of the game X_i.

An edge from node S_1 to node S_2 in the diagram indicates that the move S_2 can be played if S_1 was the last moved played. It is labelled by the name of the strategy that is responsible of making the move or by 'Env.' to denote a move played by the environment (*i.e.*, the opponent in the overall game $[\![\Gamma \rightarrow o]\!]$). For instance the edge B_k, ◑ $\xrightarrow{ev}$ B_0, ◐ tells us that if B_k, ◑ is the last move played then the evaluation strategy can respond with the move B_k, ◑. The game starts at node C, $\bullet$ which corresponds to the initial move of the overall game. The dashed-edges correspond to moves played by the copy-cat strategies π_i and ev.



where $k \in \{1..p\}$, $i, j \in \{1..n\}$ and $p \geq 1$.

Figure 4.2: Flow-diagram for interaction plays of $\langle\!\langle \Gamma \vdash x_i N_1 \ldots N_p : o \rangle\!\rangle$.

We observe that every (superficial) internal move played in some component B_k for $k \in \{0..p\}$ is either a copy of a previous external move, or it is subsequently copied to a external component by the copy-cat strategy ev or π_i: ◐-moves from B_0 are copies by ev of O-moves from C and ◑-moves from B_k, $k \in \{1..p\}$; ◑-moves from B_0 are copies by π_i of O-moves from X_i; ◐-moves from B_k, $k \in \{1..p\}$ are copies by ev of ◑-moves from the components B'_k of B_0; and finally ◑-moves from B_k, $k \in \{1..p\}$ are copied into B_0.

Moreover, each move on the diagram of Fig. 4.2 has either a single outgoing copy-cat edge—in which case the following move is uniquely determined—or it has multiple out-going edges all labelled by Σ—in which case the strategy Σ determines which moves will be played next. Hence for every two consecutive moves in a play of $\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}}$ we can uniquely recover all the internal moves occurring between the two moves in the corresponding play of $\Sigma_s \| ev$ by following the arrows of the flow diagram. This transformation is called the ***syntactical uncovering function*** with respect to Σ_s and ev and is denoted $\curlyvee_{\Sigma,ev} : \Sigma_s ;^{\emptyset,\{1..p\}} ev \rightarrow \Sigma_s \| ev$. By definition it satisfies the following property:

$$\mathsf{hide}(\curlyvee_{\Sigma,ev}(u), \{1..p\}, \{0\}) = u$$

for all $u \in \Sigma_s ;^{\emptyset,\{1..p\}} ev$ whose last occurrence is an external move (*i.e.*, in C or X_i for $i \in \{1..n\}$).

Recovering the fully-revealed semantics from the syntactically-revealed semantics Given a term-in-context $\Gamma \vdash M : A$, its syntactically-revealed denotation $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle_{\mathsf{s}}$ can be obtained from $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle$ by recursively hiding the appropriate internal moves. Conversely, the fully-revealed denotation $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle$ can be obtained from $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle_{\mathsf{s}}$ by recursively applying the syntactical-uncovering transformation described in the previous paragraph for every subterm of the form $y_i N_1 \ldots N_p$.

4.2.1.5 Revealed semantics versus standard game semantics

In the standard semantics, given two strategies $\sigma : A \to B$, $\tau : B \to C$ and a sequence $s \in \sigma;\tau$, it is possible to (uniquely) recover from the sequence s the internal moves that were hidden during composition [HO00, part II]. This gives another way to compute the revealed denotation of a term: given a term M, first compute its standard game denotation. Its revealed denotation can then be obtained by recursively uncovering the internal moves for every application occurring in the term.

Conversely, the standard denotation can be obtained from the revealed denotation by filtering out all the internal moves:

$$\llbracket \Gamma \vdash M : T \rrbracket = \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle \upharpoonright \llbracket \Gamma \to T \rrbracket \ . \tag{4.7}$$

This equality remains valid if we replace the fully revealed denotation by the syntactically-revealed denotation.

Observe that the two sets of plays $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle$ and $\llbracket \Gamma \vdash M : T \rrbracket$ are not in bijection. Indeed, by definition the revealed denotation is prefix-closed therefore it also contains plays ending with an internal move. Thus the revealed denotation contains more plays than the standard denotation. What we can say, however, is that the set of plays $\llbracket \Gamma \vdash M : T \rrbracket$ is in bijection with the subset of $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle$ consisting of plays ending with an external move. Furthermore the set of complete plays of $\llbracket \Gamma \vdash M : T \rrbracket$ is in bijection with the set of complete interaction plays of $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle$.

4.2.1.6 Projection

The projection operation for justified sequences of moves of an interaction strategies (Def. 4.71) proceeds by eliminating some of the moves from the sequence. In general when projecting a sequence $s \in \Sigma$ on a subtree T', for some subtree $\Sigma' : T'$ of $\Sigma : T$, the resulting sequence is not necessarily an *interaction position* of Σ' because some internal moves may be missing from s. The following lemma shows that for strategies that are fully-revealed denotations, the projection operation generates valid positions of its sub-interaction strategies.

Lemma 4.77 (Projection for fully-revealed denotations)**.** *Let $\Sigma : T$ be a fully-revealed denotation (i.e., $\Sigma = \langle\!\langle M \rangle\!\rangle$ for some term M). Then for every sub-tree $\Sigma' : T'$ of $\Sigma : T$ and $u \in \Sigma$:*

- *if T' is the first subtree of a ';'-node in T then for every initial $\llbracket type(T') \rrbracket$-move b occurring in u we have $u \upharpoonright T' \upharpoonright b \in \Sigma'$;*
- *otherwise (T' is the subtree of a 'Λ'-node, '$\langle \_, \_ \rangle$'-node or the l^{th} subtree of a ';'-node for $l > 1$) then $u \upharpoonright T' \in \Sigma'$.*

Proof. The proof is by induction on the distance between T' and T's root. The sequence $u \upharpoonright T'$ equals $u \upharpoonright T_0 \upharpoonright \ldots \upharpoonright T_k$ for some $k \geq 0$ where the T_is are the unique subtrees of T such that $T_0 = T$, $T_k = T'$, and T_i is an immediate subtree of T_{i-1} for $1 \leq i \leq k$. Let $\Sigma_i : T_i$ denote the strategy corresponding to each subtree T_i of T. We proceed by induction on $k \geq 0$. The base case is trivial. Step case: Suppose that $v = u \upharpoonright T_{k-1} \in \Sigma_{k-1}$. We do a case analysis on the type of the root node of Σ_{k-1}. The cases 'Λ' and '$\langle \_, \_ \rangle$' are trivial. The only other possible case is '$\|$' (since Σ is a fully-revealed denotation). The result then follows by definition of $\|$ with a subtlety in the case $l = 1$: we have $\Sigma_{k-1} = \Sigma' \| \Sigma_r$, $\Sigma' : {T'}^{A \to B}$ for some strategy $\Sigma_r : T_r^{B \to C}$. When calculating the positions of the composition $\Sigma' \| \Sigma_r$, links going from initial A-moves to initial B-moves in the positions of Σ' are changed into links pointing to initial C-moves in $\Sigma' \| \Sigma_r$. Thus in order to obtain a valid position of Σ' from v we need to recover the pointers accordingly. This is precisely what the filtering operation $\_ \upharpoonright T'$ does (see Def. 4.71): if an A-move in v loses its pointer in $v \upharpoonright M_{T'}$ then its justifier in $v \upharpoonright T'$ is set to the only initial B-move b occurring in the P-view $\ulcorner v \upharpoonright M_{T'} \upharpoonright \llbracket type(T') \rrbracket \urcorner$. Hence the justification pointers are properly restored and $v \upharpoonright T' \upharpoonright b$ is indeed an uncovered position of Σ'. $\square$

Together with Lemma 4.73 this further implies:

Lemma 4.78. *Let $\Sigma = \langle\!\langle M \rangle\!\rangle : T$. For every $u \in \Sigma$ and sub-tree $\Sigma' : T'$ of $\Sigma : T$ inducing a standard strategy $\sigma' : [\![type(T')]\!]$:*

- *if T' is the first subtree of a '$;$'-node in T then for every initial $[\![type(T')]\!]$-move b occurring in u we have $u \upharpoonright [\![type(T')]\!] \upharpoonright b \in \sigma'$;*
- *otherwise (T' is the subtree of a 'Λ'-node, '$\langle \_, \_ \rangle$'-node or the l^{th} subtree of a '$;$'-node for $l > 1$) then $u \upharpoonright [\![type(T')]\!] \in \sigma'$.*

Proof. Follows immediately from Lemma 4.77 and 4.73. □

Lemma 4.79 (Well-bracketing)**.** *Let $\Sigma : T$ be the fully-revealed denotation of some term M. Then for every sub-revealed strategy $\Sigma' : T'$ of $\Sigma : T$, the standard strategy $\sigma' : [\![type(T')]\!]$ induced by Σ' is well-bracketed.*

Proof. The leaves of a fully-revealed denotation are annotated by well-bracketed strategies therefore since well-bracketing is preserved by pairing, currying and composition, all the standard strategies induced by the sub-revealed strategies of Σ are also well-bracketed. □

Lemma 4.80 (Complete interaction play)**.** *Let $\Sigma : T$ and $\Sigma_s : T$ denote respectively the fully-revealed strategy and syntactically-revealed denotation of some term (i.e., $\Sigma = \langle\!\langle M \rangle\!\rangle$ and $\Sigma_s = \langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$ for some term M). Then:*

(i) For every $u \in \Sigma$, if $u \upharpoonright [\![type(T)]\!]$ is complete (i.e., maximal and all question moves are answered) then so is u.

(ii) For every $u \in \Sigma_s$, if $u \upharpoonright [\![type(T)]\!]$ is complete then so is u.

Proof. (i) We show the contrapositive. If u is not complete then it contains an answered move b. If b is not internal then it appears in $u \upharpoonright [\![type(T)]\!]$ and therefore $u \upharpoonright [\![type(T)]\!]$ is not complete. Otherwise, let $\Sigma' : T'$ be the subtree of Σ where the internal move b is uncovered: Σ' is of the form $\Sigma_1 ;^{S,P} \Sigma_2$ for some $S, P \subseteq \mathbb{N}$ with $\Sigma_1 : \langle\!\langle T_1^{A \to B} \rangle\!\rangle$ and $\Sigma_2 : \langle\!\langle T_2^{B \to C} \rangle\!\rangle$, and b belongs to some uncovered component of B (*i.e.*, whose index is in S).

Since b is unanswered in u, it is not answered in $u \upharpoonright A, B$ and $u \upharpoonright B, C$ either; thus the sequences $u \upharpoonright A, B$ and $u \upharpoonright B, C$ are not complete. This further implies that $u \upharpoonright A, C$ is not complete (By contradiction: otherwise we would have $u \upharpoonright A \to C = q\, u'\, a$ for some initial question q and answer a; but since q and a both belong to C this implies $u \upharpoonright B \to C = q \ldots a$). By Lemma 4.78, $u \upharpoonright B \to C$ belongs to the standard strategy induced by Σ_2, and by Lemma 4.79 this strategy is well-bracketed, thus $u \upharpoonright B \to C$ is well-bracketed and since its first question is answered it is necessarily complete.

We have shown that $u \upharpoonright [\![A \to C]\!] = u \upharpoonright [\![type(T')]\!]$ is not complete. We then conclude by observing that if $u \upharpoonright [\![type(T')]\!]$ is not complete for some sub-tree T' of T then $u \upharpoonright [\![type(T)]\!]$ is not complete either. This can be shown by an easy induction on the distance between the root of T' and T: The currying and pairing cases are trivial; for the composition case, the argument is similar to the one used in the previous paragraph.

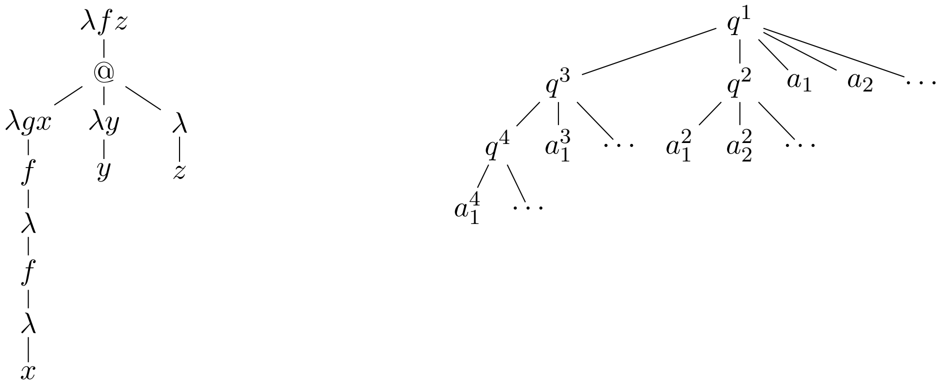
(ii) By applying the syntactical uncovering function on u we obtain a position v of Σ satisfying $u \upharpoonright [\![type(T)]\!] = v \upharpoonright [\![type(T)]\!]$. Hence by (i), v is complete, and therefore so is u (since u is the subsequence of v obtained by recursively hiding internal moves). □

4.2.2 Relating computation trees and games

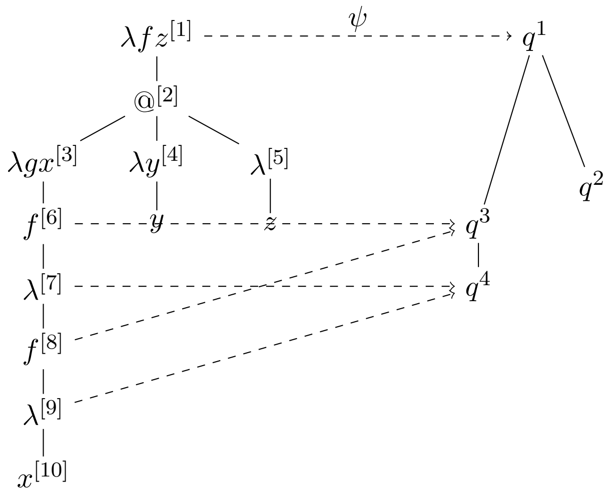
In this paragraph we relate nodes of the computation tree to moves of the game arena. First we use an example to explain the insight before giving the formal definition.

4.2.2.1 Example

Consider the following term $M \equiv \lambda f^{o\to o} z^o.(\lambda g^{o\to o} x^o.f(fx))(\lambda y^o.y)z$ of type $(o \to o) \to o \to o$. Its η-long normal form is $\lambda f^{o\to o} z^o.(\lambda g^{o\to o} x^o.f(fx))(\lambda y^o.y)(\lambda.z)$. The following figure represents side-by-side the computation tree of M (left) and the arena of the game $[\![(o \to o) \to o \to o]\!]$ (right):



Now consider the following partial mapping ψ (represented by a dashed line in the diagram below) from the set of nodes of the computation tree to the set of moves in the arena: (For simplicity, we now omit answer moves when representing arenas.)



Consider the justified sequence of moves:

$$s = q^1\ q^3\ q^4\ q^3\ q^4\ q^2 \in [\![M]\!]\ .$$

Its image by ψ gives a justified sequence of nodes of the computation tree:

$$r = \lambda fz \cdot f^{[6]} \cdot \lambda^{[7]} \cdot f^{[8]} \cdot \lambda^{[9]} \cdot z$$

where $s_i = \psi(r_i)$ for all $i < |s|$.

The sequence r is in fact the core of the following traversal:

$$t = \lambda fz \cdot @^{[2]} \cdot \lambda gx^{[3]} \cdot f^{[6]} \cdot \lambda^{[7]} \cdot f^{[8]} \cdot \lambda^{[9]} \cdot x^{[10]} \cdot \lambda^{[5]} \cdot z\ .$$

This example motivates the next section where we formally define the mapping ψ for any given simply-typed term.

4.2.2.2 Formal definition

We now establish formally the relationship between games and computation trees. We assume that a term $\Gamma \vdash M : T$ in η-long normal form is given.

NOTATIONS 4.81 We suppose that computation tree $\tau(M)$ is given by a pair (N, E) where N is the set of nodes and $E \subseteq N \times N$ is the parent-child relation. We have $N = IN \cup L$ where IN and L are the set of inner nodes and leaf nodes respectively. Let $\mathcal{D}$ be the set of values of the base type o. If n is an inner node in IN then the value-leaves attached to the node n are written v_n where v ranges in $\mathcal{D}$. Similarly, if q is a question in A then the answer moves enabled by q are written v_q where v ranges in $\mathcal{D}$.

Definition 4.82 (Mapping from nodes to moves of the standard game semantics).

- Let n be a node in $IN_\lambda \cup IN_{\mathsf{var}}$ and q be a question move of some game A such that n and q are of type $(A_1, \ldots, A_p, o)$ for some $p \geq 0$. Let $\{q^1, \ldots, q^p\}$ (resp. $\{v_q \mid v \in \mathcal{D}\}$) be the set of question-moves (resp. answer-moves) enabled by q in A (each q^i being of type A_i).

  We define the function $\psi_A^{n,q}$ from $N^{n\vdash}$— nodes that are hereditarily enabled by n—to moves of A as:

$$\begin{aligned}\psi_A^{n,q} \quad &= \quad \{n \mapsto q\} \cup \{v_n \mapsto v_q \mid v \in \mathcal{D}\} \\ &\cup \begin{cases} \bigcup_{m \in IN_{\mathsf{var}} \mid n \vdash_i m} \psi_A^{m,q^i}, & \text{if } n \in IN_\lambda \ ; \\ \bigcup_{i=1..p} \psi_A^{n.i,q^i}, & \text{if } n \in IN_{\mathsf{var}} \ . \end{cases}\end{aligned}$$

- Suppose $\Gamma = x_1 : X_1, \ldots, x_k : X_k$. Let q_0 denote $[\![\Gamma \rightarrow T]\!]$'s initial move[2] and suppose that the set of moves enabled by q_0 in $[\![\Gamma \rightarrow T]\!]$ is $\{q_{x_1}, \ldots, q_{x_k}, q^1, \ldots, q^p\} \cup \{v_q \mid v \in \mathcal{D}\}$ where each q^i is of type A_i and q_{x_j} of type X_j.

  We define $\psi_M : N^{\circledast\vdash} \rightarrow [\![\Gamma \rightarrow T]\!]$ (or just ψ if there is no ambiguity) as:

$$\begin{aligned}\psi_M = \quad & \{r \mapsto q_0\} \cup \{v_r \mapsto v_{q_0} \mid v \in \mathcal{D}\} \\ \cup \quad & \bigcup_{n \in IN_{\mathsf{var}} \mid \circledast \vdash_i n} \psi_{[\![\Gamma \rightarrow T]\!]}^{n,q^i} \\ \cup \quad & \bigcup_{n \in IN_{\mathsf{fv}} \mid n \text{ labelled } x_j, j \in \{1..k\}} \psi_{[\![\Gamma \rightarrow T]\!]}^{n,q_{x_j}} \ .\end{aligned}$$

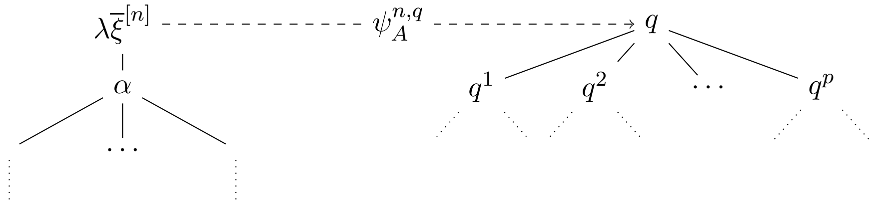
It can easily be checked that the domain of definition of $\psi_A^{n,q}$ is indeed the set of nodes that are hereditarily enabled by n and similarly, the domain of ψ_M is the set of nodes that are hereditarily enabled by the root (this includes free variable nodes and nodes that are hereditarily enabled by free variable nodes). Also, if M is closed then we have $\psi_M = \psi_{[\![\rightarrow T]\!]}^{\circledast,q_0}$.

The construction of the function $\psi_A^{n,q}$, defined above, goes as follows. Let p be the arity of the type of n and q.

- If $p = 0$ then n is a dummy λ-node or a ground type variable: $\psi_A^{n,q}$ maps n to the initial move q.

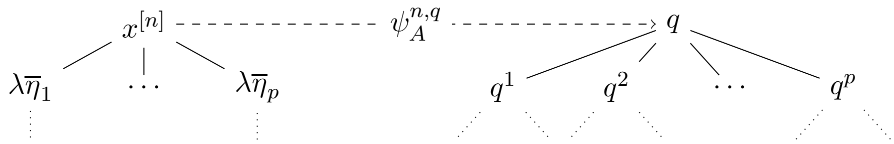
- If $p \geq 1$ and $n \in IN_\lambda$ with n labelled $\lambda\overline{\xi} = \lambda\xi_1 \ldots \xi_p$ then the sub-computation tree rooted at n and the arena A have the following forms (value-leaves and answer moves are not represented for simplicity):

[2] Arenas involved in the game semantics of simply-typed lambda calculus are trees: they have a single initial move.



For each abstracted variable ξ_i there exists a corresponding question move q^i of the same order in the arena. The function $\psi_A^{n,q}$ maps each free occurrence of ξ_i in the computation tree to the move q^i.

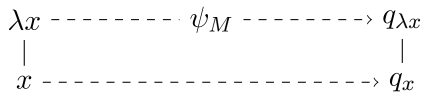
- If $p \geq 1$ and $n \in IN_{\mathsf{var}}$ then n is labelled with a variable $x : (A_1, \ldots, A_p, o)$ with children nodes $\lambda\overline{\eta}_1$, ..., $\lambda\overline{\eta}_p$. The computation tree $\tau(M)$ rooted at n and the arena A have the following forms:



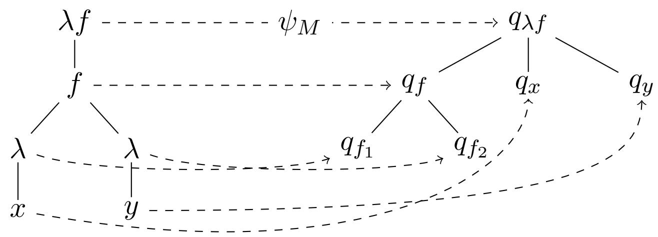
and $\psi_A^{n,q}$ maps each node $\lambda\overline{\eta}_i$ to the question move q^i.

Example 4.83. For each of the following examples of term-in-context $\Gamma \vdash M : T$, we represent the computation tree $\tau(M)$, the arena of the game $[\![\Gamma \to T]\!]$, and the function ψ_M (in dashed lines):

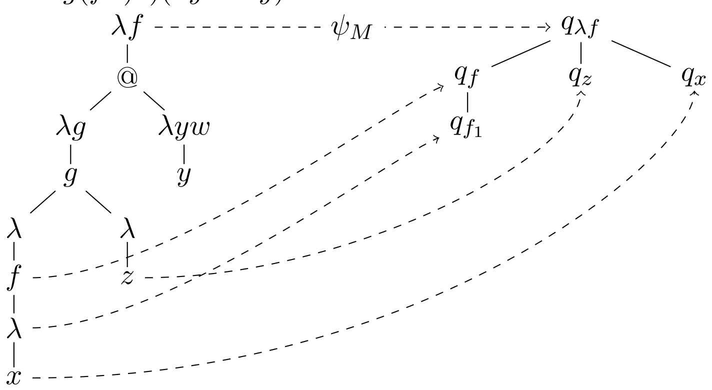
- $M \equiv \lambda x^o.x$



- $M \equiv \lambda f^{(o,o,o)}.fxy$



- $M \equiv \lambda f^{(o,o)}.(\lambda g^{(o,o,o)}.g(fx)z)(\lambda y^o w^o.y)$



Lemma 4.84.

(i) ψ_M maps λ-nodes to O-questions, variable nodes to P-questions, value-leaves of λ-nodes to P-answers and value-leaves of variable nodes to O-answers;

(ii) *ψ_M preserves hereditary enabling: a node $n \in N^{\circledast\vdash}$ is hereditarily enabled by some node $n' \in N^{\circledast\vdash}$ in $\tau(M)$ if and only if the move $\psi_M(n)$ is hereditarily enabled by $\psi_M(n')$ in $[\![\Gamma \rightarrow T]\!]$;*

(iii) *ψ_M maps a node of a given order to a move of the same order;*

(iv) *Let $s \in \mathcal{T}rav(M)^{\upharpoonright\circledast}$. The P-view (resp. O-view) of $\psi_M(s)$ and s are computed identically (i.e., the set of positions of occurrences that need to be deleted in order to obtain the P-view (resp. O-view) is the same for both sequences).*

Proof. (i), (ii) and (iii) are direct consequences of the definition. (iv): Because of (i) and since t and $\psi_M(t)$ have the same pointers, the computations of the views of the sequence of moves and the views of the sequence of nodes follow the same steps. □

The convention chosen to define the order of the root node (see Def. 4.8) permits us to have property (iii). This explains why we have defined the order of the root node differently from other lambda nodes.

By extension, we can define the function ψ_M on $\mathcal{T}rav(M)^{\upharpoonright\circledast}$, the set of traversal cores, as follows:

Definition 4.85 (Mapping traversal cores to sequences of moves)**.** The function ψ_M maps any traversal core $u = u_0 u_1 \ldots \in \mathcal{T}rav(M)^{\upharpoonright\circledast}$ to the following justified sequence of moves of the arena $[\![\Gamma \rightarrow T]\!]$: $\psi_M(u) = \psi_M(u_0)\ \psi_M(u_1)\ \psi_M(u_2) \ldots$ where $\psi_M(u)$ is equipped with u's pointers.

The pointer-free function underlying ψ_M is thus a monoid homomorphism.

4.2.3 Mapping traversals to interaction plays

Let I be the interaction game of the revealed strategy $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ and M_I be the set of equivalence classes of moves from $\mathcal{M}_I$.

Let r be a lambda node in IN_{spawn} (the children nodes of @/Σ-nodes). We write $\Gamma(r) \vdash M^{(r)} : T(r)$ to denote the subterm of $\lceil M \rceil$ rooted at r (thus $\Gamma(r) \subseteq \Gamma$). We consider the function $\psi_{M^{(r)}}$ which maps nodes of $N^{r\vdash}$ to moves of $[\![\Gamma(r) \rightarrow T(r)]\!]$. Since $\mathcal{M}_I$ contains the moves from the standard game $[\![\Gamma(r) \rightarrow A(r)]\!]$, we can consider $\psi_{M^{(r)}}$ as a function from $N^{r\vdash}$ to $\mathcal{M}_I$.

Every node in $n \in N \setminus (N_@ \cup N_\Sigma)$ is either hereditarily enabled by the root or by some λ-node in IN_{spawn}. Therefore we can define the following relation ψ^*_M from $N \setminus (N_@ \cup N_\Sigma)$ to $\mathcal{M}_I$:

$$\psi^*_M = \psi_M \quad \cup \bigcup_{r \in IN_{\mathsf{spawn}}} \psi_{M^{(r)}} \ .$$

This relation is totally defined on $N \setminus (N_@ \cup N_\Sigma)$ since those nodes are either hereditarily justified by the root, by an @-node or by a Σ-node. Moreover it is a relation and *not* a function since for a given variable node x, for every spawn node r occurring in the path from x to $\circledast$, x is hereditarily enabled by r *with respect to the computation tree* $\tau(M^{(r)})$. Thus the domains of definition of the relations $\psi_{M^{(r)}}$ for such nodes r overlap. It can be easily check, however, that for every node $n \in N \setminus (N_@ \cup N_\Sigma)$, the moves in $\psi^*_M(n)$ are all $\sim$-equivalent, which leads us to the following definition:

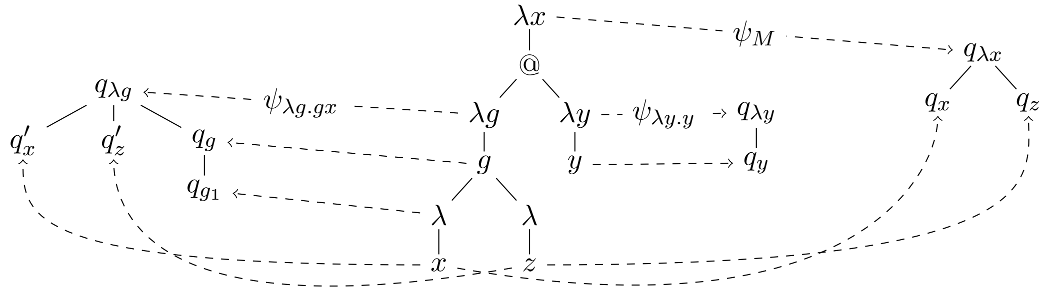
Definition 4.86 (Mapping from nodes to moves of the syntactically-revealed semantics)**.** We define the *function* $\varphi_M : N \setminus (N_@ \cup N_\Sigma) \rightarrow M_I$ as follows: For $n \in N \setminus (N_@ \cup N_\Sigma)$, $\varphi_M(n)$ is defined as the $\sim$-equivalence class containing the set $\psi^*_M(n)$. We omit the subscript in φ_M if there is no ambiguity.

Definition 4.87 (Mapping sequences of nodes to sequences of moves)**.** We define the function φ_M from $\mathcal{T}rav(M)^\star$ to justified sequence of moves in M_I as follows. If $u = u_0 u_1 \ldots \in \mathcal{T}rav(M)^\star$ then:

$$\varphi_M(s) = \varphi_M(u_0)\ \varphi_M(u_1)\ \varphi_M(u_2) \ldots$$

where $\varphi_M(u)$ is equipped with u's pointers.

Example 4.88. Take $M \equiv \lambda x^o.(\lambda g^{(o,o)}.gxz)(\lambda y^o.y)$. The diagram below represents the computation tree (middle) and the relation $\psi^*_M = \psi_{\lambda x} \cup \psi_{\lambda g.gx} \cup \psi_{\lambda y.y}$ (dashed-lines).



where $q'_x \sim q_x$, $q'_z \sim q_z$, $q_g \sim q_{\lambda y}$, $q_{g_1} \sim q_y$ and $q_{\lambda g} \sim q_{\lambda x}$.

Lemma 4.89 (Traversal projection lemma)**.** *Let $\Delta \vdash Q : A$ be a subterm of $\lceil M \rceil$ and $\circledast_Q$ denote the root lambda node of the subtree of $\tau(M)$ corresponding to the term Q. Let $t \in \mathcal{T}rav(M)$, r_0 be an occurrence of $\circledast_Q$ in t and m_0 be the occurrence of the initial A-move $\varphi_M(r_0)$ in $\varphi_M(t^\star)$. Then:*

$$\varphi_Q(t^\star \upharpoonright N^{(\circledast_Q)} \upharpoonright r_0) = \varphi_M(t^\star) \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle \upharpoonright m_0 \ .$$

Proof. Firstly we observe that the expression "$\varphi_Q(t^\star \upharpoonright N^{(\circledast_Q)} \upharpoonright r_0)$" is well-defined. Indeed, by Proposition 4.62 $t \upharpoonright\!\upharpoonright r_0$ is a traversal of $\mathcal{T}rav(Q)$ therefore the sequence $t^\star \upharpoonright N^{(\circledast_Q)} \upharpoonright r_0$, which is equal to $(t \upharpoonright\!\upharpoonright r_0)^\star$ by Lemma 4.63, does belong to $\mathcal{T}rav(Q)^\star$.

We now make the assumption that $\circledast_Q$ is a level-2 lambda nodes (*i.e.*, a grand-child of the root $\circledast$). The proof easily generalizes to other lambda nodes by iterating the argument at every lambda node occurring in the path from $\circledast_Q$ to $\circledast$.

Claim: (i) The set of occurrence positions of $t^\star$ that are removed by the operation $\_ \upharpoonright N^{(\circledast_Q)}$ is the same as the set of positions of $\varphi_M(t^\star)$ removed by the operation $\_ \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle$. (ii) The justification pointers in the sequences of nodes $t^\star \upharpoonright N^{(\circledast_Q)}$ are the same as those of the sequence of moves $\varphi_M(t^\star) \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle$.

Indeed: (i) follows from the fact that, by definition, the range of the function φ_M restricted to $N^{(\circledast_Q)}$ is included in $M_{\langle\!\langle \Delta \rightarrow A \rangle\!\rangle}$ (the set of moves of the interaction game of Q).

(ii) By Def. 4.87, the sequences $\varphi_M(t^\star)$ and $t^\star$ have the same justification pointers. The projections $\_ \upharpoonright N^{(\circledast_Q)}$ and $\_ \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle$ both alter the pointers in the sequences $\varphi_M(t^\star)$ and $t^\star$, but they do so identically: the operation $\_ \upharpoonright N^{(\circledast_Q)}$ (Def. 4.44) alters pointers only for variable nodes that are free in $N^{(\circledast_Q)}$; it makes them point to the only occurrence of $\circledast_Q$ in the P-view at that point (which is also the only occurrence of a level-2 lambda node in the P-view). Similarly, the operation $\_ \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle$ (Def. 4.71) alters pointers only for initial A-moves: it makes them point to the only occurrence of an initial B-move in the P-view at that point. Further φ_M maps free variables in $N^{(\circledast_Q)}$ to initial A-moves, and level-2 lambda nodes to initial B-moves.

Hence the claim holds which subsequently implies $\varphi_M(t^\star) \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle = \varphi_M(t^\star \upharpoonright N^{(\circledast_Q)})$. Thus $\varphi_M(t^\star) \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle \upharpoonright m_0 = \varphi_M(t^\star \upharpoonright N^{(\circledast_Q)}) \upharpoonright m_0 = \varphi_M(t^\star \upharpoonright N^{(\circledast_Q)} \upharpoonright r_0)$. Finally, since the function φ is defined inductively on the structure of the computation tree, the restriction of φ_M to $N^{\circledast_Q}$ coincides with φ_Q. ◻

The following lemma states that projecting the image of a traversal by φ gives the image of the traversal's core:

Lemma 4.90 (Core projection lemma)**.**

$$\varphi_M(\mathcal{T}rav(M)^\star) \upharpoonright [\![\Gamma \rightarrow T]\!] = \psi_M(\mathcal{T}rav(M)^{\upharpoonright \circledast}) \ .$$

Proof. Let H be the set of nodes of $\tau(M)$ which are mapped by $\psi^*(M)$ to moves that are $\sim$-equivalent to moves in $[\![\Gamma \rightarrow T]\!]$. We need to show that $H = N^{\circledast\vdash}$.

Since $\psi_M \subseteq \psi^*(M)$ and the image of $\psi(M)$ is $[\![\Gamma \rightarrow T]\!]$, H must contain the domain of $\psi(M)$ which is precisely $N^{\circledast\vdash}$. Conversely, suppose that a node $n \in N \setminus (N_@ \cup N_\Sigma)$ is mapped by $\varphi^*(M)$ to some move $m \in \mathcal{M}_I$ which is $\sim$-equivalent to some move in $[\![\Gamma \rightarrow T]\!]$. If $m = \psi_M(n)$ then $n \in N^{\circledast\vdash}$. Otherwise, $m = \psi_{\kappa(\odot)}(n)$ for some $\odot \in IN_{\mathsf{spawn}}$. There may be several nodes $\odot$ such that n belongs to the domain of definition of $\psi_{M^{(\odot)}}$, w.l.o.g. we can take $\odot$ to be the one which is closest to the root. Let $\Gamma(\odot) \vdash M^{(\odot)} : T(\odot)$. Suppose that m is $\sim$-equivalent to a move from

- the subgame $[\![\Gamma]\!]$ of $[\![\Gamma \rightarrow T]\!]$, then this means that n is hereditarily justified by a free variable node in M and therefore $n \in N^{\circledast\vdash}$.
- the subgame $[\![T]\!]$ of $[\![\Gamma \rightarrow T]\!]$ then m must belong to the subgame $\Gamma(\odot)$ of $[\![\Gamma(\odot) \rightarrow T(\odot)]\!]$. Indeed, since $\odot$'s parent node is an application node, moves in the subgame $[\![T(\odot)]\!]$ correspond to internal moves of the application. By definition of the interaction strategy for the application case, such moves can only be $\sim$-equivalent to other internal moves and thus cannot be equivalent to a move from $[\![T]\!]$.
  Consequently, n is hereditarily justified by a free variable node z in $M^{(\odot)}$. By assumption, $\odot$ is the closest node to the root $\circledast$ (excluding $\circledast$ itself) for which n belongs to $N^{\odot\vdash}$ (the domain of definition of $\psi_{M^{(\odot)}}$). Hence z is not bound by any λ-node occurring in the path to the root. Thus $z \in N^{\circledast\vdash}$ and therefore $n \in N^{\circledast\vdash}$.

Hence $H = N^{\circledast\vdash}$. Consequently, for every traversal t we have $\varphi_M(t^\star) \upharpoonright [\![\Gamma \rightarrow T]\!] = \varphi_M(t^\star \upharpoonright N^{\circledast\vdash})$ which equals $\varphi_M(t \upharpoonright \circledast)$ by Lemma 4.43. ∎

4.2.4 The correspondence theorem for the pure simply-typed lambda calculus

In this section, we establish a connection between the revealed semantics of a simply-typed term without interpreted constants (*i.e.*, $\Sigma = \emptyset$) and the traversals of its computation tree: we show that the set $\mathcal{T}rav(M)$ of traversals of the computation tree is isomorphic to the set of uncovered plays of the strategy denotation (this is the counterpart of Ong's "Path-Traversal Correspondence" Theorem [Ong06a]), and that the set of traversal cores is isomorphic to the strategy denotation.

Preliminary lemmas

Notation 4.91 For every node occurrence n in a justified sequence (of nodes or of moves) u we write $\mathsf{ptrdist}_u(n)$, or just $\mathsf{ptrdist}(n)$ if there is no ambiguity, to denote the distance between n and its justifier in u if it has one, and 0 otherwise.

Lemma 4.92.

$$\left(\begin{array}{l} t \cdot n_1, t \cdot n_2 \in \mathcal{T}rav(M) \\ \wedge \ \ n_1 \neq n_2 \end{array} \right) \implies n_1, n_2 \in N^{\circledast\vdash}_\lambda \wedge (\psi(n_1) \neq \psi(n_2) \vee \mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)) \ .$$

Proof. Take $t \cdot n_1, t \cdot n_2 \in \mathcal{T}rav(M)$. Suppose that n_1 and n_2 belong to two distinct categories of nodes (IN_{var}, $IN_@$, IN_λ, IN_Σ, L_{var}, $L_@$, L_λ, or L_Σ) then necessarily one must be visited with the rule (InputVar) and the other by (InputVal)—they are the only rules with a common domain of definition—thus one is a leaf-node and the other is an inner node which implies that $\psi(n_1) \neq \psi(n_2)$. Otherwise n_1 and n_2 belong to the same category of nodes and we proceed by case analysis:

- If $n_1, n_2 \in IN_@$ then $t \cdot n_1$ and $t \cdot n_2$ are formed using the (App) rule. Since this rule is deterministic we must have $n_1 = n_2$ which violates the second hypothesis.

- If $n_1, n_2 \in L_@$ then the traversals are formed using the deterministic rule ($\mathsf{Value}^{@\mapsto\lambda}$) which again violates the second hypothesis.
- If $n_1, n_2 \in IN_\Sigma$ then they are formed using a deterministic constant rule (see Def. 4.25).
- If $n_1, n_2 \in L_\Sigma$ then they are formed using a deterministic value-constant rule.
- If $n_1, n_2 \in IN_{\mathsf{var}}$ then $t \cdot n_1$ and $t \cdot n_2$ were formed using either rule (Lam) or (App). But these two rules are deterministic and their domains of definition are disjoint. Hence again the second hypothesis is violated.
- If $n_1, n_2 \in L_{\mathsf{var}}$ then either the traversals were both formed using the deterministic rule ($\mathsf{Value}^{\mathsf{var}\mapsto\lambda}$) in which case the second hypothesis is violated; or they were formed with ($\mathsf{InputValue}$) in which case n_1 and n_2 are two different value leaves belonging to $N^{\circledast\vdash}_\lambda$ and justified by the same input variable node. Thus by definition of ψ, $\psi(n_1) \neq \psi(n_2)$.
- If $n_1, n_2 \in IN_\lambda$ then the traversals $t \cdot n_1$ and $t \cdot n_2$ must have been formed using either rule (Root), (App), (Var) or ($\mathsf{InputVar}$). Since all these rules have disjoint domains of definition, the same rule must have been use to form $t \cdot n_1$ and $t \cdot n_2$. But since the rules (Root), (App) and (Var) are all deterministic, the rule used is necessarily ($\mathsf{InputVar}$).
  By definition of ($\mathsf{InputVar}$), $n_1, n_2 \in IN^{\circledast\vdash}_\lambda$, the parent node of n_1 and the parent node of n_2 all occur in $\llcorner t_{\leqslant x} \lrcorner$ where $x \in IN^{\circledast\vdash}_{\mathsf{var}}$ denotes the pending node at t. If n_1 and n_2 have the same parent node in $\tau(M)$ then since $n_1 \neq n_2$, by definition of ψ, $\psi(n_1) \neq \psi(n_2)$. If their parent node is different, then n_1 and n_2 are necessarily justified by two different occurrences in t therefore $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$.
- If $n_1, n_2 \in L_\lambda$ then either the traversals $t \cdot n_1$ and $t \cdot n_2$ were formed using ($\mathsf{Value}^{\lambda\mapsto\mathsf{var}}$) or they were formed with ($\mathsf{Value}^{\lambda\mapsto @}$) but this is impossible since these two rules are deterministic and $n_1 \neq n_2$. $\square$

The function φ_M regarded as a function from the set of nodes $N \setminus N_@$ of the computation tree to moves in arenas is not injective. (For instance the two occurrences of x in the computation tree of $\lambda f x.fxx$ are mapped to the same question move.) However the function φ_M defined on the set of @-free traversals is injective, and similarly the function ψ_M defined on the set of traversal cores is injective as the following lemma shows:

Lemma 4.93 (ψ_M and φ_M are injective)**.** *For every two traversals t_1 and t_2:*

(i) If $\varphi(t_1^\star) = \varphi(t_2^\star)$ then $t_1^\star = t_2^\star$;

(ii) if $\psi(t_1 \upharpoonright \circledast) = \psi(t_2 \upharpoonright \circledast)$ then $t_1 \upharpoonright \circledast = t_2 \upharpoonright \circledast$.

Proof. (i) The result is trivial if either t_1 or t_2 is empty. Otherwise, suppose that $t_1^\star \neq t_2^\star$ then necessarily $t_1 \neq t_2$. W.l.o.g. we can assume that the two traversals differ only by their last node (or last node's pointer). Thus we have $t_1 = t \cdot n_1$ and $t_2 = t \cdot n_2$ for some sequence t and some occurrences n_1, n_2 where either n_1 and n_2 are two distinct nodes in the computation tree or $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$.

If $n_1 = n_2$ and $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$ then n_1, n_2 are not @-nodes nor Σ-nodes (since for such nodes we would have $\mathsf{ptrdist}(n_1) = 0 = \mathsf{ptrdist}(n_2)$). By definition of the sequence $\varphi(t_1)$ we have $\mathsf{ptrdist}(\varphi(n_1)) = \mathsf{ptrdist}(n_1)$ and similarly $\mathsf{ptrdist}(\varphi(n_2)) = \mathsf{ptrdist}(n_2)$ thus $\varphi(t' \cdot n_1) \neq \varphi(t' \cdot n_2)$. Finally since $n_1, n_2 \notin (IN_@ \cup IN_\Sigma)$ we also have $\varphi((t' \cdot n_1)^\star) \neq \varphi((t' \cdot n_2)^\star)$. Hence $\varphi(t_1^\star) \neq \varphi(t_2^\star)$.

If $n_1 \neq n_2$ then by Lemma 4.92 n_1, n_2 are not @-nodes or Σ-nodes (since such nodes are not hereditarily justified by the root) and we have either $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$ or $\varphi(n_1) = \psi(n_1) \neq \psi(n_2) = \varphi(n_2)$. Hence $\varphi(t_1^\star) \neq \varphi(t_2^\star)$.

(ii) Suppose that $t_1 \upharpoonright \circledast \neq t_2 \upharpoonright \circledast$ then necessarily $t_1 \neq t_2$. W.l.o.g. we can assume that the two sequences differ only by their last occurrence. Hence we have $t_1 = t \cdot n_1$, $t_2 = t' \cdot n_2$ for some sequence t and some nodes n_1, n_2 where either $n_1 \neq n_2$ or $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$.

If $n_1 \neq n_2$ then Lemma 4.92 gives $\psi(t_1 \upharpoonright \circledast) \neq \psi(t_2 \upharpoonright \circledast)$. Otherwise $n_1 = n_2$ and $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$. The only rules that can visit the same node with two different pointers are $(\mathsf{InputVar})$ and $(\mathsf{InputValue})$, thus n_1 and n_2 must be in $N_\lambda^{\circledast\vdash}$. Hence:

$$\psi(t_i \upharpoonright \circledast) = \psi(t \upharpoonright \circledast) \cdot \psi(n_i) \text{ for } i \in \{1..2\}$$

where $\mathsf{ptrdist}_{\psi(t_i \upharpoonright r)}(\psi(n_i)) = \mathsf{ptrdist}_{t_i \upharpoonright r}(n_i)$.

Furthermore, since $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$ and $t_{1<n_1} = t_{2<n_2}$ we have $\mathsf{ptrdist}_{t_1 \upharpoonright \circledast}(n_1) \neq \mathsf{ptrdist}_{t_2 \upharpoonright \circledast}(n_2)$. Thus $\psi(t_1 \upharpoonright \circledast) \neq \psi(t_2 \upharpoonright \circledast)$. □

Corollary 4.94.

(i) φ *defines a bijection from* $\mathcal{T}rav(M)^\star$ *to* $\varphi(\mathcal{T}rav(M)^\star)$ *;*

(ii) ψ *defines a bijection from* $\mathcal{T}rav(M)^{\upharpoonright\circledast}$ *to* $\psi(\mathcal{T}rav(M)^{\upharpoonright\circledast})$ *.*

The following lemma says that extending a traversal locally also extends the traversal globally: the traversal t of M can be extended by extending a sub-traversal t' of some subterm of M. This is not obvious since t' is a subsequence of t which means that the nodes in t' are also present in t with the same pointers but with some other nodes interleaved in between. However these interleaved nodes are inserted in a way that allows us to apply on t the rule that was used to extend the sub-traversal t':

Lemma 4.95 (Sub-traversal progression)**.** *Let* $\circledast_j$ *be a lambda node in* $\tau(M)$*,* $t = t' \cdot t^\omega$ *be a justified sequence of nodes of* $\tau(M)$*, and* r_j *be an occurrence of* $\circledast_j$ *in* t *different from* t^ω*. If*

1. t' *is a traversal of* $\tau(M)$*,*

2. t^ω *appears in* $t \upharpoonleft\!\upharpoonright r_j$*,*

3. $t \upharpoonleft\!\upharpoonright r_j$ *is a traversal of* $\tau(M^{(\circledast_j)})$ *and its last node is visited using a rule different from* $(\mathsf{InputVar})$ *and* $(\mathsf{InputVar}^{\mathsf{val}})$*,*

then t *is a traversal of* $\tau(M)$*.*

Proof. Let $t_j = t \upharpoonleft\!\upharpoonright r_j$. Since t' is a traversal of M, by Prop. 4.62 the sequence $t' \upharpoonleft\!\upharpoonright r_j$ (which is also the immediate prefix of t_j) is a traversal of $\tau(M^{(\circledast_j)})$. We proceed by case analysis on the last rule used to produce the traversal t_j and we show that t is a traversal of M:

- (Empty), (Root). These cases do not occur since $|t_j| \geq 2$. Indeed, t_j contains at least t^ω and r_j which are two different occurrences.
- (Lam) We have $t_j = \ldots \cdot \lambda\overline{\xi} \cdot n$. Since $t_j \sqsubseteq t$, the node $\lambda\overline{\xi}$ also occurs in t. Therefore using the rule (Lam) in M we can form the traversal $t_{\leqslant \lambda\overline{\xi}} \cdot n$. But then we have $(t_{\leqslant \lambda\overline{\xi}} \cdot n) \upharpoonleft\!\upharpoonright r_j = t_{\leqslant \lambda\overline{\xi}} \upharpoonleft\!\upharpoonright r_j \cdot n = t_{j \leqslant \lambda\overline{\xi}} \cdot n = t_j = t \upharpoonleft\!\upharpoonright r_j$. Thus, since t's last node and n both appear in $t \upharpoonleft\!\upharpoonright r_j$, this implies that $t_{\leqslant \lambda\overline{\xi}} \cdot n = t$. Hence t is a traversal of M.
- (App) $t_j = \ldots \cdot \lambda\overline{\xi} \cdot @ \cdot n$. The same reasoning as in the previous case permits us to conclude.
- $(\mathsf{Value}^{@\mapsto\lambda})$ $t_j = \ldots \cdot \lambda\overline{\xi} \cdot @ \ldots v_@ \cdot v_{\lambda\overline{\xi}}$. Since $t_j \sqsubseteq t$, the nodes $\lambda\overline{\xi}$, $@$, $v_@$ and $v_{\lambda\overline{\xi}}$ all appear in t. Moreover, since $\lambda\overline{\xi}$ is a lambda node appearing in $t \upharpoonleft\!\upharpoonright r_j$, its immediate successor must also appear in $t \upharpoonleft\!\upharpoonright r_j$. Thus the two nodes $\lambda\overline{\xi}$ and $@$ are also consecutive in t. Hence we can use the rule $(\mathsf{Value}^{@\mapsto\lambda})$ in the computation tree $\tau(M)$ to produce the traversal $t_{\leqslant v_{\lambda\overline{\xi}}} \cdot n$ and by the same reasoning as in the previous case, we conclude that necessarily $t = t_{\leqslant v_{\lambda\overline{\xi}}} \cdot n$.
- $(\mathsf{Value}^{\mathsf{var}\mapsto\lambda})$ $t_j = \ldots \cdot \lambda\overline{\xi} \cdot x \ldots v_x \cdot v_{\lambda\overline{\xi}}$. This case is identical to the previous case.
- $(\mathsf{Value}^{\lambda\mapsto@})$ $t_j = \ldots \cdot @ \cdot \lambda\overline{z} \ldots v_{\lambda\overline{z}} \cdot v_@$. Same as in the previous case by observing that $@$ and $\lambda\overline{z}$ are necessarily consecutive in t.
- $(\mathsf{InputValue})$ and $(\mathsf{InputVar})$. By assumption these cases do not happen.

- (Var) $t_j = \ldots \cdot p \cdot \lambda\overline{x} \ldots x_i \cdot \lambda\overline{\eta_i}$ for some variable $x_i \in IN^{@\vdash}_{\mathsf{var}}$.

In general, two nodes p and $\lambda\overline{x}$ appearing consecutively in t_j are not necessarily consecutive in t, because a traversal of M, can "jump" from p to a node outside of the subterm $M^{(\circledast_j)}$, and thus not appearing in $t_j = t \upharpoonright\!\upharpoonright r_j$. But this situation cannot happen here. Indeed, suppose that $t_{\leqslant p}$ extends to $t_{\leqslant p} \cdot m$ in $\tau(M)$. In t_j, all the nodes from the thread of $\lambda\overline{\eta_i}$ are hereditarily justified by the same initial @-node which necessarily occurs after r_j—the first occurrence of t_j. Consequently p belongs to $IN^{@\vdash}_{\mathsf{var}}$ and therefore the traversal $t_{\leqslant p} \cdot m$ must have been formed using the rule (Var) in $\tau(M)$. Since p appears in $t \upharpoonright\!\upharpoonright r_j$, by Lemma 4.52(i), all the nodes in the thread of p in t appear in $t \upharpoonright\!\upharpoonright r_j$. Thus m appears in $t \upharpoonright\!\upharpoonright r_j$ (since by O-visibility it points in the thread of p). Hence we have $(t_{\leqslant p} \cdot m) \upharpoonright\!\upharpoonright r_0 = t_{<p} \upharpoonright\!\upharpoonright r_0 \cdot p \cdot m$ which implies that m is precisely the occurrence $\lambda\overline{x}$.
Hence the nodes p, $\lambda\overline{x}$, x_i and $\lambda\overline{\eta_i}$ all appear in t with the two nodes p and $\lambda\overline{x}$ appearing consecutively. We can therefore use the rule (Var) in M to form the traversal t.

- (Value$^{\lambda\mapsto\mathsf{var}}$) Same proof as in the previous case.
- (Σ)/(Σ-var) Same as (App) and (Var).
- (Σ-Value) Same as (Value$^{\lambda\mapsto\mathsf{var}}$). □

The correspondence theorem

We now state and prove the correspondence theorem for the simply-typed lambda calculus without interpreted constants ($\Sigma = \emptyset$). This theorem establishes a correspondence between the denotation of a term in the *intensional* game model and the set of traversals of its computation tree. The result extends immediately to the simply-typed lambda calculus with *uninterpreted* constants since we can regard constants as being free variables.

Theorem 4.96 (The Correspondence Theorem)**.** *For every simply-typed term $\Gamma \vdash M : T$, φ_M defines a bijection from $\mathcal{T}rav(M)^\star$ to $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ and ψ_M defines a bijection from $\mathcal{T}rav(M)^{\upharpoonright\circledast}$ to $[\![\Gamma \vdash M : T]\!]$:*

$$\begin{aligned}
\varphi_M &: \mathcal{T}rav(\Gamma \vdash M : T)^\star \xrightarrow{\cong} \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}} \\
\psi_M &: \mathcal{T}rav(\Gamma \vdash M : T)^{\upharpoonright\circledast} \xrightarrow{\cong} [\![\Gamma \vdash M : T]\!] \ .
\end{aligned}$$

Remark 4.97 By Corollary 4.94, we just need to show that φ_M and ψ_M are *surjective*, that is to say: $\varphi_M(\mathcal{T}rav(M)^\star) = \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ and $\psi_M(\mathcal{T}rav(M)^{\upharpoonright\circledast}) = [\![\Gamma \vdash M : T]\!]$. Moreover the former implies the latter, indeed:

$$\begin{aligned}
[\![\Gamma \vdash M : T]\!] &= \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}} \upharpoonright [\![\Gamma \to T]\!] && \text{by (4.7) from Sec. 4.2.1.5} \\
&= \varphi_M(\mathcal{T}rav(M)^\star) \upharpoonright [\![\Gamma \to T]\!] && \text{by assumption} \\
&= \psi_M(\mathcal{T}rav(M)^{\upharpoonright\circledast}) && \text{by Lemma 4.90;}
\end{aligned}$$

therefore we just need to prove that $\varphi_M(\mathcal{T}rav(M)^\star) = \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$.

The proof is rather technical, we first give an overview of the argument: We proceed by induction on the structure of the computation tree. The variable and abstraction cases are trivial. The guts of the proof lies in the application case. There are two subcases depending on whether the operator is a variable or an abstraction. Consider the latter case; the computation tree $\tau(M)$ has the following form:

$$\begin{array}{ccc}
 & \lambda\overline{\xi} & \\
 & | & \\
 & @ & \\
\diagup & | & \diagdown \\
\tau(N_0) & \ldots & \tau(N_p)
\end{array}$$

A traversal of $\tau(M)$ goes as follows: It starts at the root $\lambda\overline{\xi}$ of the tree $\tau(M)$ (rule (Root)), visits the node @ (rule (Lam)) and the root of $\tau(N_0)$ (rule (App)) and then proceeds by traversing the subtree $\tau(N_0)$. While doing so, some variable y_i bound by $\tau(N_0)$'s root may be reached, in which case the traversal is interrupted by a jump to $\tau(N_i)$'s root (performed with the rule (Var)) and the process goes on with $\tau(N_i)$. Again, if the traversal encounters a variable bound by $\tau(N_i)$'s root then the traversal of $\tau(N_i)$ is interrupted and the traversal of $\tau(N_0)$ resumes. This schema is repeated until the traversal of $\tau(N_0)$ is completed[3].

A traversal of M consists therefore of an initialization part followed by an interleaving of a traversal of N_0 and several traversals of N_i for $i = 1..p$. This schema is reminiscent of the way the evaluation copy-cat map ev works in game semantics. The crucial point of the proof is that every jump of the traversal from one subterm to another is permitted by one of the "copy-cat" rules (Var) or (Value). We show by a second induction that these copy-cat rules implement precisely the copy-cat evaluation strategy ev.

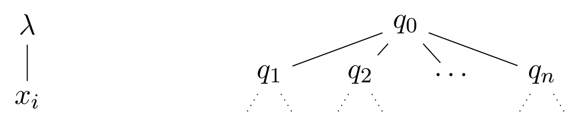
Proof. Let $\Gamma \vdash M : T$ be a simply-typed term where $\Gamma = x_1 : X_1, \ldots x_n : X_n$. We assume that M is already in η-long normal form. By remark 4.97 we just need to show that $\varphi_M(\mathcal{T}rav(M)^\star) = \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$. We proceed by induction on the structure of M:

- (abstraction) $M \equiv \lambda\overline{\xi}.N : \overline{Y} \to B$ where $\overline{\xi} = \xi_1 : Y_1, \ldots \xi_n : Y_n$. On the one hand we have:

$$\begin{array}{rcl} \langle\!\langle \Gamma \vdash \lambda\overline{\xi}.N : T \rangle\!\rangle_{\mathsf{s}} & = & \Lambda^n(\langle\!\langle \overline{\xi}, \Gamma \vdash N : B \rangle\!\rangle_{\mathsf{s}}) \\ & \simeq & \langle\!\langle \overline{\xi}, \Gamma \vdash N : B \rangle\!\rangle_{\mathsf{s}} \ . \end{array}$$

  On the other hand, the computation tree $\tau(N)$ is isomorphic to $\tau(\lambda\overline{\xi}.N)$ (up to renaming of the computation tree's root), and $\mathcal{T}rav(N)$ is isomorphic to $\mathcal{T}rav(\lambda\overline{\xi}.N)$. Hence we can conclude using the induction hypothesis.

- (variable) $M \equiv x_i$. Since M is in η-long normal form, x must be of ground type. The computation tree $\tau(M)$ and the arena $\langle\!\langle \Gamma \to o \rangle\!\rangle_{\mathsf{s}}$ are represented below (value leaves and answer moves are not represented):



  Let π_i denote the i^{th} projection of the interaction game semantics. We have:

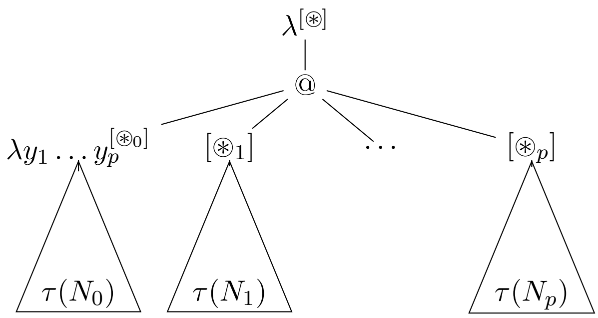
$$\langle\!\langle M \rangle\!\rangle_{\mathsf{s}} = \pi_i = \mathsf{Pref}(\{q_0 \cdot q^i \cdot v_{q^i} \cdot v_{q_0} \mid v \in \mathcal{D}\}) \ .$$

  It is easy to see that traversals of M are precisely the prefixes of $\lambda \cdot x_i \cdot v_{x_i} \cdot v_\lambda$. Since M is in β-normal we have $\mathcal{T}rav(M)^\star = \mathcal{T}rav(M)$, and since $\varphi_M(\lambda) = q_0$ and $\varphi_M(x_i) = q^i$ we have:

$$\varphi_M(\mathcal{T}rav(M)^\star) = \varphi_M(\mathcal{T}rav(M)) = \varphi_M(\mathsf{Pref}(\lambda \cdot x_i \cdot v_{x_i} \cdot v_\lambda)) = \langle\!\langle M \rangle\!\rangle_{\mathsf{s}} \ .$$

- (@-application) $M \equiv N_0 N_1 \ldots N_p : o$ where N_0 is not a variable. We have the typing judgements $\Gamma \vdash N_0 N_1 \ldots N_p : o$ and $\Gamma \vdash N_i : B_i$ for $i \in 0..p$ where $B_0 = (B_1, \ldots, B_p, o)$ and $p \geq 1$. The tree $\tau(M)$ has the following form:

[3] Since we are considering simply-typed terms, the traversal does indeed terminate. However this will not be true anymore in the PCF case.



where $\circledast_j$ denotes the root of $\tau(N_j)$ for $j \in \{0..p\}$. We have:

$$\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}} = \underbrace{\langle \langle\!\langle \Gamma \vdash N_0 : B_0 \rangle\!\rangle_{\mathsf{s}}, \ldots, \langle\!\langle \Gamma \vdash N_p : B_p \rangle\!\rangle_{\mathsf{s}} \rangle}_{\Sigma} \parallel ev \ .$$

We now use the notations introduced in Fig. 4.1 from section 4.2.1.3 which fixes the names of the different games involved in the interaction strategy $\langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$. In particular the games A, B and C are defined as:

$$\begin{aligned} A &= X_1 \times \ldots \times X_n \\ B &= \underbrace{((B'_1 \times \ldots \times B'_p) \rightarrow o')}_{B_0} \times B_1 \times \ldots \times B_p \\ C &= o \ . \end{aligned}$$

Let q_0 and q'_0 be the initial question of C and B_0 respectively.

$\subseteq$ We first prove that $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}} \subseteq \varphi_M(\mathcal{T}rav(M)^\star)$. Suppose $u \in \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$. We give a constructive proof that there is a traversal t such that $\varphi_M(t^\star) = u$ by induction on u.

For the base case $u = \epsilon$, take t to be the empty traversal formed with (Empty). *Step case:* Suppose that $u = u' \cdot m \in \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ for some move $m \in M_T$ where $u' = \varphi_M(t'^\star)$ for some traversal t' of $\tau(M)$. By unraveling the definition of $u \in \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ we have:

$$\left.\begin{array}{ll} (a) & u \in J_T \ ; \\ (b) & \text{For every occurrence } b \text{ in } u \text{ of an initial } B_k\text{-move, for some } k \in \{0..p\}\text{:} \\ & \left\{ \begin{array}{ll} u \upharpoonright T^{0k} \upharpoonright b \in \langle\!\langle N_k \rangle\!\rangle_{\mathsf{s}} \ , & \\ u \upharpoonright T^{0k'} \upharpoonright b = \epsilon & \text{for every } k' \in \{0..p\} \setminus \{k\} \ ; \end{array}\right. \\ (c) & u \upharpoonright B_0 = u \upharpoonright B_1, \ldots, B_p, C \ . \end{array}\right\} \quad (4.8)$$

We recall that each $m \in M_T$ is an equivalence class of moves from $\mathcal{M}_T$. For every game A appearing in the interaction game T we will write "$m \in A$" to mean that some element of the class m belongs to the set of moves M_A. Similarly, for every sub-interaction game T' of T, we write "$m \in T'$" to mean that some element of the class m belongs to the set of moves $\mathcal{M}_{T'}$. We proceed by case analysis on m: We either have $m \in C$ or $m \in T^0$; in the last case m is either in A, a superficial internal move in B or a profound internal move in B:

- Suppose $m \in C$. Moves in C are played by the standard strategy ev that does not contain any internal move. Hence m is either q_0 or v_{q_0} for some $v \in \mathcal{D}$.
  Suppose that $m = q_0$. Since q_0 can occur only once in u we have $u = q_0$ and the traversal $t = \lambda^{[\circledast]}$ formed with (Root) clearly satisfies $\varphi(t^\star) = u$.
  Otherwise $m = v_{q_0}$. This P-move is played by the copy-cat strategy ev therefore it is the copy of some answer $v_{q'_0}$ to the question q'_0 from the sub-game o'. The move $v_{q'_0}$

is necessarily the immediate predecessor of m in u. (Indeed the play $u_{\leqslant v_{q'_0}} \upharpoonright A, B$ is complete since its first move q'_0 is answered by $v_{q'_0}$, and therefore $u_{\leqslant v_{q'_0}} \upharpoonright T^0$ is also complete by Lemma 4.80; thus no profound internal move can be played between $v_{q'_0}$ and v_{q_0}, and therefore these two moves are consecutive.)
Hence by the induction hypothesis the last move in t' is $\varphi(v_{q'_0}) = v_{\lambda y_1}$. The rules ($\mathsf{Value}^{\lambda\mapsto @}$) and ($\mathsf{Value}^{@\mapsto\lambda}$) permits us to extend the traversal t' to $t' \cdot v_@ \cdot v_{\lambda\overline{\xi}}$ where $v_@$ and $v_{\lambda\overline{\xi}}$ point to the second and first node of t' respectively. Clearly we have $\varphi_M((t' \cdot v_@ \cdot v_{\lambda\overline{\xi}})^\star) = u$.

- Suppose $m \in T^0$ and m is an initial move in B_0. Then necessarily m is $q'_0 \in [\![o']\!]$, the copy-cat move of the initial move $q_0 \in C$ of u. Hence $u = q_0 \cdot q'_0$. The rules (Root), (App) and (Lam) permit us to build the traversal $t = \lambda^{[\circledast]} \cdot @ \cdot \lambda\overline{y}^{[\circledast_0]}$ which clearly satisfies $\varphi_M(t^\star) = u$.
- Suppose $m \in T^0$ and m is an initial move in B_k for some $k \in \{1..p\}$. Then m is necessarily a copy-cat move played by the evaluation strategy, and the move m^1 immediately preceding m in u is an initial move of the component B'_k of B_0. Thus since $\varphi_M(t'^\omega) = m^1$, t'^ω must be an occurrence of the node y_k—the k^{th} variable bound by $\lambda\overline{y}$. We can thus form, with the rule (Var), the traversal $t = t' \cdot \circledast_k$ satisfying $\varphi_M(t^\star) = \varphi_M(t'^\star) \cdot m = u$.
- Suppose $m \in T^0$ and m is not initial in B. In $u \upharpoonright T^0$, m must be hereditarily justified by some initial move b in B_k for some $k \in \{0..p\}$. Since $u \upharpoonright T^{0k} \upharpoonright b \in \langle\!\langle N_k \rangle\!\rangle_{\mathsf{s}}$, the outermost induction hypothesis gives us:

$$u \upharpoonright T^{0k} \upharpoonright b = \varphi_{N_k}(t_k^\star) \tag{4.9}$$

for some traversal $t_k \in \mathcal{T}rav(N_k)$ where w.l.o.g. we can assume that $t_k^\omega \notin N_@$. We have:

$$\begin{aligned}
\varphi_M(t_k^\omega) &= (\varphi_M(t_k^\star))^\omega && \text{since } t_k^\omega \notin N_@ \\
&= ((u' \cdot m) \upharpoonright T^{0k} \upharpoonright b)^\omega && \text{by (4.9)} \\
&= ((u' \upharpoonright T^{0k} \upharpoonright b) \cdot m))^\omega && \text{since } m \text{ is h.j. by } b \text{ and belongs to } T^{0k} \\
&= m \ .
\end{aligned}$$

Take $t = t' \cdot t_k^\omega$ where t_k^ω points in t' to the image by φ_M of the occurrence justifying m in u. Since $t_k^\omega \neq @$ we have $t^\star = t'^\star \cdot t_k^\omega$ where t_k^ω justifier in $t'^\star$ is the same as its justifier in t.
Hence we have $\varphi_M(t^\star) = \varphi_M(t'^\star) \cdot \varphi_M(t_k^\omega)$ which, by the innermost I.H. together with the previous equation, equals $u' \cdot m$ where m's justifier in u' corresponds to $\varphi_M(t_k^\omega)$'s justifier in $\varphi_M(t'^\star)$. Consequently:

$$\varphi_M(t^\star) = u \ . \tag{4.10}$$

We are half-done at this point, it remains to show that t is indeed a traversal of $\tau(M)$. Let r_k denote the occurrence of the root $\circledast_k$ in t that is mapped to the occurrence b in $\varphi_M(t^\star)$. We make the following claim:

$$t_k = t \upharpoonright\!\upharpoonright r_k \ . \tag{4.11}$$

Indeed we have:

$$\begin{aligned}
\varphi_{N_k}(t_k^\star) &= u \upharpoonright T^{0k} \upharpoonright b && \text{by (4.9)} \\
&= \varphi_M(t^\star) \upharpoonright T^{0k} \upharpoonright b && \text{by (4.10)}
\end{aligned}$$

$$= \varphi_{N_k}(t^\star \upharpoonright N^{(\circledast_k)} \upharpoonright r_k) \qquad \text{by Lemma 4.89.}$$

Since φ_{N_k} is a bijection from $\mathcal{T}rav(N_k)^\star$ to $\varphi_{N_k}(\mathcal{T}rav(N_k)^\star)$ (by Corollary 4.94) this implies that $t_k^\star = t^\star \upharpoonright N^{(\circledast_k)} \upharpoonright r_k$ which in turn equals $(t \upharpoonright\!\upharpoonright r_k)^\star$ by Lemma 4.63 from Sec. 4.1.3.6. But since t_k and t do not end with an @-node, this implies equality (4.11).

We now show that t is indeed a traversal by a case analysis of the rule used to visit the last occurrence of t_k in the tree $\tau(N_k)$:

(a) Suppose the rule used to visit t_k^ω is neither (InputVar) nor (InputVar$^{\mathsf{val}}$). Then by Lemma 4.95, t is a traversal of M.

(b) Suppose t_k^ω is visited with (InputVar). Then t_k is of the form

$$t_k = \ldots \cdot z \cdot \ldots \cdot t_k^\omega$$

for some input-variable $z \in IN_{\mathsf{var}}^{\circledast_k \vdash}$ occurring in $\llcorner t_k \lrcorner$ and where $t_k^\omega \in IN_\lambda^{\circledast_k \vdash}$. Thus:

$$u = \ldots \cdot \underbrace{\psi_{N_k}(z)}_{=m^3} \cdot \ldots \cdot \underbrace{\psi_{N_k}(t_k^\omega)}_{=m} \ .$$

The occurrence t_k^ω is hereditarily enabled by the root $\circledast_k$ itself enabled by an application node, thus t_k^ω is not hereditarily enabled by the root $\circledast$. Since only nodes that are hereditarily enabled by the root are mapped to move in A we know that $\psi_{N_k}(t_k^\omega)$ is not played in A and therefore $\psi_{N_k}(t_k^\omega) \in B_k$. Similarly we have $\psi_{N_k}(z) \in B_k$.

Now consider the top-most composition in the interaction strategy $\langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$—that of the interaction strategy $\Sigma : A \to B$ with the evaluation copy-cat strategy $ev : B \to o$. Consider the sub-sequence $u \upharpoonright A, B, C$ of u consisting only of moves involved in this top-most composition (*i.e.*, the internal moves coming from other compositions at deeper level in the revealed semantics are removed). Since z is a variable node, the move $m^3 = \psi_{N_k}(z) \in B_k$ is a P-move with *respect to the game* $[\![A \to B_k]\!]$, and therefore it is an O-move in the game $[\![B \to o]\!]$. Consequently the strategy ev is responsible to play at $u_{\leqslant m^3} \upharpoonright A, B, C$. Let m^2 denote the move played by ev which immediately follows m^1 in $u \upharpoonright A, B, C$.

We claim that m^3 and m^2 are also consecutive in u. That is to say that no internal moves generated from the other compositions at deeper levels in the interaction strategy can ever be played between m^3 and m^2. Indeed, firstly the strategy ev is a pure standard strategy thus it does not play any profound internal move. Furthermore, suppose that the strategy Σ comes from the composition $\Sigma_l \| \Sigma_r$ of two interaction strategies $\Sigma_l : A \to D$ and $\Sigma_r : D \to B$ for some game D, then by the Switching Condition for function-space game [HO00] the Opponent cannot switch of component, and thus the move following m^3 in the interaction sequence $u \upharpoonright A, D, B$ must belong to B. Hence internal moves from D cannot be played immediately after m^3.

Similarly, we can show that the move m is played by the strategy ev and is the copy of the move m^1 immediately preceding it in $u \upharpoonright A, B, C$ as well as in u.

Hence the sequence u has the following form:

$$u = \ldots \cdot m^3 \cdot m^2 \cdot \ldots \cdot m^1 \cdot m \ .$$

Consequently we have:

$$t_k = \ldots \cdot z \cdot \ldots \cdot t_k^\omega \qquad\qquad t' = \ldots \cdot z \cdot \lambda\overline{y} \cdot \ldots \cdot y \ .$$

The first equation implies that t_k^ω is the i^{th} child of z in the computation tree, thus since $z \notin IN^{\circledast\vdash}$, we can apply the (Var) rule to the second equation which produces the traversal of $\tau(M)$:

$$t' \cdot t_k^\omega = \ldots \cdot z \cdot \lambda\overline{y} \cdot \ldots \cdot y \cdot t_k^\omega$$

which is precisely the sequence t. Hence t is indeed a traversal of $\tau(M)$. The diagram on Fig. 4.3 shows an example of such interaction sequence u.

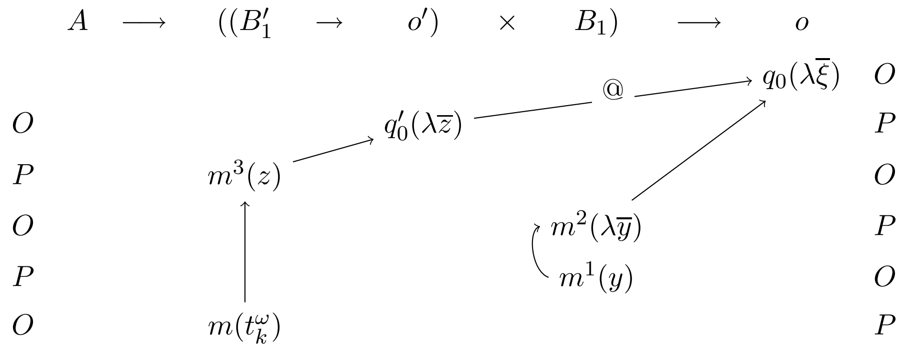


Figure 4.3: Example of a sequence $u \upharpoonright A, B, C$ for $u \in \langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$ and $l = 1$.

(c) Suppose t_k's last move is visited with the rule ($\mathsf{InputVar}^{\mathsf{val}}$) then the proof is the same as in the previous case but with ($\mathsf{InputVar}^{\mathsf{val}}$) substituted for ($\mathsf{InputVar}$).

$\supseteq$ The converse, $\varphi_M(\mathcal{T}rav(M)^\star) \subseteq \langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$, is the easy part of the proof.
Let u be as sequence of $\varphi_M(\mathcal{T}rav(M)^\star)$. Then $u = \varphi_M(t^\star)$ for some traversal t of $\tau(M)$. To show that u is a position of $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ we have to prove that it satisfies the three conditions of (4.8):

- (a) By definition, φ_M maps justified sequences of nodes to justified sequences of moves from M_T therefore $\varphi_M(t^\star) \in J_T$.
- (b) Take an initial B-move $b \in B_k$, for some $k \in \{0..p\}$, occurring in $\varphi_M(t^\star)$. There is a corresponding occurrence r_k in t of a level-2 lambda node $\circledast_k$ of $\tau(M)$. By definition, the function φ_M maps nodes from the subtree of $\tau(M)$ rooted at $\circledast_{k'}$, for every $k' \in \{0..p\}$, to moves of the game $\langle\!\langle \Gamma \rightarrow B_{k'} \rangle\!\rangle_{\mathsf{s}}$ that are hereditarily justified by some occurrence of $\varphi_M(\circledast_{k'})$. Hence for every $k' \in \{0..p\} \setminus \{k\}$ we clearly have $\varphi_M(t^\star) \upharpoonright T^{0k'} \upharpoonright b = \epsilon$. Moreover:

$$\begin{aligned}
u \upharpoonright T^{0k} \upharpoonright b &= \varphi_M(t^\star) \upharpoonright T^{0k} \upharpoonright b \\
&= \varphi_M(t^\star \upharpoonright N^{(\circledast_k)} \upharpoonright r_k) && \text{by Lemma 4.89} \\
&= \varphi_M((t \upharpoonright\!\!\upharpoonright r_k)^\star) && \text{by Lemma 4.63} \\
&= \varphi_{N_k}((t \upharpoonright\!\!\upharpoonright r_k)^\star) && \text{since } t \upharpoonright\!\!\upharpoonright r_k \text{ is a traversal of } N_k \text{ by Prop. 4.62} \\
&\in \varphi_{N_k}(\mathcal{T}rav(N_k)^\star) \\
&= \langle\!\langle N_k \rangle\!\rangle_{\mathsf{s}} && \text{by the induction hypothesis.}
\end{aligned}$$

- (c) We can show that $\varphi_M(t^\star) \upharpoonright B_0 = \varphi_M(t^\star) \upharpoonright B_1, \ldots, B_p, C$ by a trivial induction on the traversal t. (This property holds because of the way the traversal rules mimic the behaviour of the evaluation strategy.)

- (Var-application) $M \equiv x_i N_1 \ldots N_p : o$.
  The revealed denotation is $\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}} = \underbrace{\langle \pi_i, \langle\!\langle \Gamma \vdash N_1 : B_1 \rangle\!\rangle_{\mathsf{s}}, \ldots, \langle\!\langle \Gamma \vdash N_p : B_p \rangle\!\rangle_{\mathsf{s}} \rangle}_{\Sigma} ;^{\emptyset,\{1..p\}} ev$

  and the computation tree is

$$\begin{array}{ccc} & \lambda^{[\circledast]} & \\ & | & \\ & x_i & \\ \swarrow & | & \searrow \\ \tau(N_1)^{[\circledast_1]} & \ldots & \tau(N_p)^{[\circledast_p]} \end{array} .$$

We use the notations of Fig. 4.1 for names of the games involved in the interaction strategy. The composition of Σ with ev takes place on the following games:

$$\overbrace{X_1 \times \ldots \overbrace{((B''_1 \times \ldots \times B''_p) \to o'')}^{X_i} \ldots \times X_n}^{A} \xrightarrow{\Sigma} \overbrace{\boxed{\overbrace{((B'_1 \times \ldots \times B'_p) \to o')}^{B_0} \times B_1 \times \ldots B_p}}^{B} \xrightarrow{ev} \overbrace{o}^{C}$$

Let q_0, q'_0 and q''_0 be the initial question of C, B_0 and X_i respectively.

$\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}} \subseteq \varphi_M(\mathcal{T}rav(M)^\star)$. We show (constructively) by induction that for every $v \in \Sigma \| ev$ there is some traversal t such that the sequence $u = \mathsf{hide}(v, \{0..p\}, \{0\})$ equals $\varphi_M(t^\star)$. The base case $v = \epsilon$ is trivial. Suppose that $v = v' \cdot m \in \Sigma \| ev$ where $\mathsf{hide}(v', \{0..p\}, \{0\}) = \varphi_M(t'^\star)$ for some traversal t' of $\tau(M)$ and move $m \in M_T$. Unraveling the definition of $v \in \Sigma \| ev$ gives

$$\left.\begin{array}{l} \text{- } v \in J_T; \\ \text{- for every occurrence } b \text{ in } v \text{ of an initial } B_k\text{-move for some } k \in \{0..p\}: \\ \quad v \upharpoonright T^{00} \upharpoonright b \in \pi_i \text{ if } k = 0 \text{ and } v \upharpoonright T^{0k} \upharpoonright b \in \langle\!\langle N_k \rangle\!\rangle_{\mathsf{s}} \text{ if } k > 0, \\ \quad \text{and } \forall k' \in \{0..p\} \setminus \{k\}.\ v \upharpoonright T^{0k'} \upharpoonright b = \epsilon; \\ \text{- and } v \upharpoonright B_0 = v \upharpoonright B_1, \ldots, B_p, C \ . \end{array}\right\} \quad (4.12)$$

We proceed by case analysis on m. It is either played in A, B or C.

1. $m \in C$. The proof is the same as in the @-application case except that the rules ($\mathsf{Value}^{\lambda \mapsto \mathsf{var}}$) and ($\mathsf{Value}^{\mathsf{var} \mapsto \lambda}$) are used instead of ($\mathsf{Value}^{\lambda \mapsto @}$) and ($\mathsf{Value}^{@ \mapsto \lambda}$) respectively.
2. m is a superficial internal B-move. Then $\mathsf{hide}(v, \{0..p\}, \{0\}) = \mathsf{hide}(v', \{0..p\}, \{0\})$ so we can directly conclude from the I.H.
3. m is a profound internal B-move. Then necessarily m belongs to B_k for some $k \in \{1..p\}$ (since π_i does not contain internal moves). Thus m must be hereditarily justified by some $b \in B_k$. The treatment of this case is identical to the @-application case where $m \in T^0$ is not initial in B and $b \in B_k$ for some $k \in \{0..p\}$.
4. $m \in A$. Let b denote the initial B_k-move that hereditarily justifies m for some $k \in \{0..p\}$. If $k > 0$ then the treatment is the same as in case 3. Otherwise $b \in B_0$:
   - Suppose m is an occurrence of the initial o''-move q''_0. Then m is played by π_i and therefore is the copy of q'_0 itself the copy of the initial move q_0 of v. Thus $v = q_0 \cdot q'_0 \cdot q''_0$ and $u = q_0 \cdot q''_0$. The traversal $t = \lambda^{[\circledast]} \cdot x_i$ formed using the rules (Root) and (Lam) meets the requirement.
   - Otherwise since $v \upharpoonright b \in \pi_i$ we have $v \upharpoonright b \upharpoonright X_i = v \upharpoonright b \upharpoonright B_0$ therefore m is necessarily hereditarily justified by the *first* occurrence of q''_0 in v.
     * Suppose m is an $\bullet$-question. Then the preceding move in v is necessarily a $\circ$-move also played in A by the strategy π_i and therefore it is also hereditarily justified by the first occurrence of q''_0.
       By definition of φ_M, the last node in t' is a variable node (if the preceding move is a $\circ$-question) or a value-leaf of a lambda node (if the preceding move is a $\circ$-answer) that is hereditarily justified by the node x_i. Hence the rule ($\mathsf{InputVar}$) can be applied at t'.

Let m' be m's justifier in v' and α' be the corresponding node in t' that φ_M maps to m'. Suppose m is the i^{th} move enabled by m' in the arena and let α be the i^{th} child node of α' in $\tau(M)$. By definition of φ_M we have $\varphi_M(\alpha) = m$. We want to show that we can use the rule (InputVar) to append α to the traversal t'. Since we have $v \upharpoonright A, C \in [\![M]\!]$, by O-visibility m' appears in $\llcorner v' \upharpoonright A, C \lrcorner$, and by the induction hypothesis we have $v' \upharpoonright A, C = \psi_M(t' \upharpoonright r)$. Hence

$$\begin{aligned} m' \in \llcorner \psi_M(t' \upharpoonright r) \lrcorner &= \psi_M(\llcorner t' \upharpoonright r \lrcorner) \\ &= \varphi_M(\llcorner t' \upharpoonright r \lrcorner) && \text{since } \varphi_M \text{ and } \psi_M \text{ coincide on } N^{\circledast\vdash}, \\ &= \varphi_M(\llcorner t' \lrcorner) && \text{by Lemma 4.64.} \end{aligned}$$

This implies that α' appears in $\llcorner t' \lrcorner$ which allows us to use the rule (InputVar) to form the traversal $t = t' \cdot \alpha$ satisfying $\varphi_M(t^\star) = \mathsf{hide}(v, \{0..p\}, \{0\})$.

* Suppose m is a $\circ$-answer. The same argument as above holds but using (InputValue) instead of (InputVar).
* Suppose m is an $\bullet$-question. We proceed identically using the rule (Lam) instead of (InputVar). The proof that α' appears in the P-view $\ulcorner t' \urcorner$ goes as follows: Let $\ulcorner v \urcorner$ denote the *core* of the interaction sequence v [McC96b]. By P-visibility in $v \upharpoonright A, C$, m occurs in $\ulcorner v' \upharpoonright A, C \urcorner$. Further we have $\ulcorner v' \upharpoonright A, C \urcorner = \ulcorner v' \urcorner \upharpoonright A, C$ [McC96b], and clearly $\ulcorner v' \urcorner \upharpoonright A, C$ equals $\ulcorner \mathsf{hide}(v', \{0..p\}, \{0\}) \urcorner \upharpoonright A, C$. Hence

$$m' \in \ulcorner \varphi_M(t'^\star) \urcorner \upharpoonright A, C \sqsubseteq \ulcorner \varphi_M(t'^\star) \urcorner \ .$$

This implies that α' occurs in $\ulcorner t'^\star \urcorner$, which is a subsequence of $\ulcorner t' \urcorner$ by (4.1). (See Sec. 4.1.3.5).
* If m is a $\circ$-answer then we proceed as above but using the rule (Value) instead.

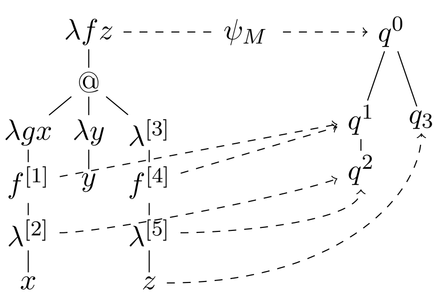
$\varphi_M(\mathcal{T}rav(M)^\star) \subseteq \langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$. Let t be some traversal of $\tau(M)$. To show that $\varphi_M(t^\star)$ is a position of $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ we have to prove that $\varphi_M(t^\star) = \mathsf{hide}(v, \{0..p\}, \{0\})$ for some v satisfying condition (4.12). It suffices to take $v = \Upsilon_{\Sigma,ev}(\varphi_M(t^\star))$ where $\Upsilon_{\Sigma,ev}$ denotes the function defined in Sec. 4.2.1.4 that transforms plays of the syntactically-revealed semantics to their corresponding plays of the fully-revealed semantics. The rest of the argument is the same as in the @-application case. □

Corollary 4.98. *If M is in β-normal form then for every traversal t, $\varphi_M(t)$ is a maximal play if and only if t is a maximal traversal.*

Proof. If M is in β-normal form then $\mathcal{T}rav(M)^{\upharpoonright\circledast} = \mathcal{T}rav(M)$ therefore φ defines a bijection on $\mathcal{T}rav(M)$. Let t be a traversal such that $\varphi(t)$ is a maximal play. Let t' be a traversal such that $t \leqslant t'$. By monotonicity of φ we have $\varphi(t) \leqslant \varphi(t')$ which implies $\varphi(t) = \varphi(t')$ by maximality of $\varphi(t)$ which in turn implies $t' = t$ by injectivity of φ. The other direction is proved identically using injectivity and monotonicity of φ^{-1}. □

The diagram on Fig. 4.4 recapitulates the main results of this section.

Example 4.99. Take $M \equiv \lambda f^{o\to o} z^o.(\lambda g^{o\to o} x^o.f x)(\lambda y^o.y)(f z) : ((o,o),o,o)$. The figure below represents the computation tree (left tree), the arena $[\![((o,o),o,o)]\!]$ (right tree) and ψ_M (dashed line). (Only question moves are shown for clarity.) The justified sequence of nodes t defined hereunder is an example of traversal:



$$t = \lambda f z \cdot @ \cdot \lambda g x \cdot f^{[1]} \cdot \lambda^{[2]} \cdot x \cdot \lambda^{[3]} \cdot f^{[4]} \cdot \lambda^{[5]} \cdot z$$

$$t \upharpoonright \lambda f z = \lambda f z \cdot f^{[1]} \cdot \lambda^{[2]} \cdot f^{[4]} \cdot \lambda^{[5]} \cdot z$$

$$\varphi_M(t \upharpoonright \lambda f z) = q^0 \cdot q^1 \cdot q^2 \cdot q^1 \cdot q^2 \cdot q^3 \in [\![M]\!] \ .$$

$$\begin{tikzcd}
 & \mathcal{T}rav(M)^\star \arrow[r, shift left, "\varphi_M"] \arrow[dl, bend left, "_ + @ + \Sigma"] & \langle\!\langle M \rangle\!\rangle_{\mathsf{s}} \arrow[l, shift left, "\varphi_M^{-1}"] \arrow[dd, shift left, "_ \upharpoonright [\![\Gamma \to T]\!]"] \\
\mathcal{T}rav(M) \arrow[ur, bend left, "_^\star = _ - @ - \Sigma"] \arrow[dr, bend right, "_ \upharpoonright \circledast"'] & & \\
 & \mathcal{T}rav(M)^{\upharpoonright \circledast} \arrow[r, shift left, "\varphi_M"] & [\![M]\!] \arrow[l, shift left, "\varphi_M^{-1}"] \arrow[uu, shift left, "\text{full uncovering}"]
\end{tikzcd}$$

where an arrow '$A \xrightarrow{f} B$' indicates that $f(A) = B$.

Figure 4.4: Transformations involved in the Correspondence Theorem.

Remark 4.100 Observe that the way we have defined traversals, the Opponent, contrary to the Proponent, is not required to play deterministically, let alone innocently. It is only required that he plays visibly (*i.e.*, his justifiers must appear in the O-view) and respects well-bracketing. This means that the game-denotation given by the Correspondence Theorem also accounts for contexts that are not simply-typed terms. This indeed corresponds to the standard innocent game model of PCF: the morphisms of the category $\mathcal{C}_{ib}$ are P-innocent strategies but not O-innocent. The addition of O-knowing-plays in the denotations is conservative for observational equivalence because the full-abstraction result holds in the category quotiented by the intrinsic preorder, and in the definition of the preorder the "test" strategy α ranges over innocent strategies only.

4.3 Extension to PCF and IA

In this section, we show how to extend the game-semantic correspondence established for the lambda calculus to other languages such as PCF and IA.

4.3.1 PCF fragment

The Y combinator needs a special treatment. In order to deal with it, we use an idea from Abramsky and McCusker's tutorial on game semantics [AM98b]: we consider the sublanguage PCF_1 of PCF in which the only allowed use of the Y combinator is in terms of the form $Y(\lambda x^A.x)$ for some type A. We will write Ω_A to denote the non-terminating term $Y(\lambda x^A.x)$ for a given type A.

We introduce the *syntactic approximants* to $\mathsf{Y}_A M$:

$$\begin{aligned}
\mathsf{Y}^0_A M &\stackrel{\text{def}}{=} \Gamma \vdash \Omega_A : A \\
\mathsf{Y}^{n+1}_A M &\stackrel{\text{def}}{=} M(\mathsf{Y}^n M) \ .
\end{aligned}$$

For every PCF term M and natural number n, we define M_n to be the PCF_1 term obtained from M by replacing each subterm of the form $\mathsf{Y}N$ with $\mathsf{Y}^n N_n$. We then have $[\![M]\!] = \bigcup_{n\in\omega} [\![M_n]\!]$ [AM98b, lemma 16].

4.3.1.1 Computation tree

In order to define the notion of computation tree for PCF terms, we first extend the inductive definition of computation tree for simply-typed terms (Def. 4.2) to PCF_1 terms by adding the new inductive case:

$$\tau(\Omega_{(A_1,\ldots,A_n,o)}) = \lambda x_1^{A_1} \ldots x_n^{A_n}.\perp$$

where $\perp$ is a special constant representing the non-terminating computation of ground type Ω_o.

We now introduce a partial order on the set of trees. A ***tree*** t is formally defined by a labelling function $t : T \to L$ where T, called the *domain* of t and written $dom(t)$, is a non-empty prefix-closed subset of some free monoid X^* and L denotes the set of possible labels. Intuitively, T represents the structure of the tree—the set of all paths—and t is the labelling function mapping paths to labels. Trees are ordered using the *approximation ordering* [KNU02, section 1]: we write $t' \sqsubseteq t$ if the tree t' is obtained from t by replacing some of its subtrees by $\perp$. Formally:

$$t' \sqsubseteq t \qquad \iff dom(t') \subseteq dom(t) \wedge \forall w \in dom(t').(t'(w) = t(w) \vee t'(w) = \perp) \ .$$

The set of all trees together with the approximation ordering form a complete partial order.

Here we take L to be the set of labels consisting of the Σ-constants, @, the special constant $\perp$, variables, and abstractions of any sequence of variables. It is easy to check that the sequence of computation trees $(\tau(M_n))_{n\in\omega}$ is a chain. We can therefore define the ***computation tree*** of a PCF term M to be the least upper-bound of the chain of computation trees of its approximants:

$$\tau(M) = \bigcup_{n\in\omega} (\tau(M_n))_{n\in\omega} \ .$$

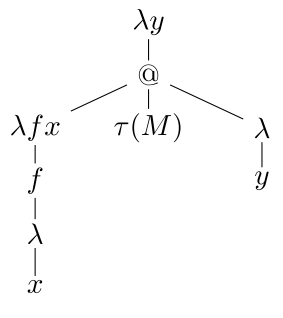
In other words, we construct the computation tree by expanding ad infinitum any subterm of the form $\mathsf{Y}M$. Thus for a term of the form $\mathsf{Y}_A F$ with $A = (A_1, \ldots, A_n, o)$, the computation tree is the unique (up to alpha-conversion) infinite tree that is solution of the equation:

$$\tau(\mathsf{Y}_A F) = \lambda \overline{x}^{\overline{A}}.\tau(F) \ \tau(\mathsf{Y}_A F) \ \tau(x_1) \ldots \tau(x_n) \tag{4.13}$$

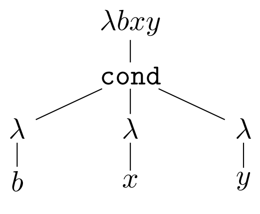
where $\overline{x} = x_1 \ldots x_n$ are fresh variables.

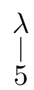
We will write $(CT, \sqsubseteq)$ to denote the set of computation trees of PCF terms ordered by the approximation ordering $\sqsubseteq$ defined above. Clearly $(CT, \sqsubseteq)$ is also a complete partial order.

Example 4.101. Take $M \equiv \mathsf{Y}(\lambda f^{(o,o)} x^o.fx)$. Its computation tree $\tau(M)$, is the tree representation of the η-long nf of the infinite term $(\lambda f^{(o,o)} x^o.fx)((\lambda f^{(o,o)} x^o.fx)((\lambda f^{(o,o)} x^o.fx)(\ldots$. It is the unique (up to alpha conversion) solution of the following equation on trees:

$\tau(M) =$ 

The remaining operators of PCF are treated as standard constants and the corresponding computation trees are constructed from the η-long normal form in the standard way. For instance the diagram below shows the computation tree for `cond` $b \ x \ y$ (left) and $\lambda x.5$ (right):





The node labelled 5 has, like any other node, children value-leaves which are not represented on the diagram above for simplicity.

4.3.1.2 Traversal

New traversal rules are added to interpret PCF constants. The arithmetic constants are traversed as follows:

- (Nat) If $t \cdot n$ is a traversal where n denotes a node labelled with some numeral constant $i \in \mathbb{N}$ then $t \cdot n \cdot i_n$ is also a traversal where i_n denotes the value-leaf of m corresponding to the value $i \in \mathbb{N}$.
- (Succ) If $t \cdot \mathtt{succ}$ is a traversal and λ denotes the only child node of $\mathtt{succ}$ then $t \cdot \mathtt{succ} \cdot \lambda$ is also a traversal.
- (Succ$'$) If $t_1 \cdot \mathtt{succ} \cdot \lambda \cdot t_2 \cdot i_\lambda$ is a traversal for $i \in \mathbb{N}$ then $t_1 \cdot \mathtt{succ} \cdot \lambda \cdot t_2 \cdot i_\lambda \cdot (i+1)_{\mathtt{succ}}$ is also a traversal.
- The rules for $\mathtt{pred}$ are defined similarly to (Succ) and (Succ$'$).

The conditional operator is implemented as follows. (We recall that a $\mathtt{cond}$-node in the computation tree has three children nodes numbered from 1 to 3 corresponding to the three parameters of the conditional operator.)

- (Cond-If) If $t_1 \cdot \mathtt{cond}$ is a traversal and λ denotes the first child of $\mathtt{cond}$ then $t_1 \cdot \mathtt{cond} \cdot \lambda$ is also a traversal.
- (Cond-ThenElse) If $t_1 \cdot \mathtt{cond} \cdot \lambda \cdot t_2 \cdot i_\lambda$ is a traversal then so is $t_1 \cdot \mathtt{cond} \cdot \lambda \cdot t_2 \cdot i_\lambda \cdot \lambda$.
- (Cond$'$) If $t_1 \cdot \mathtt{cond} \cdot t_2 \cdot \lambda \cdot t_3 \cdot i_\lambda$ is a traversal for $k = 2$ or $k = 3$ then the sequence $t_1 \cdot \mathtt{cond} \cdot t_2 \cdot \lambda \cdot t_3 \cdot i_\lambda \cdot i_{\mathtt{cond}}$ is also a traversal.

It is easy to verify that these traversal rules are all well-behaved. This completes the definition of traversals for PCF.

4.3.1.3 Revealed semantics

We recall that the definition of the syntactically-revealed semantics (Sec. 4.2.1, Def. 4.74) accounts for the presence of interpreted constants: For every Σ-constant $f : (A_1, \ldots, A_p, B)$ in the language, the revealed strategy of a term of the form $\lambda\overline{\xi}.f N_1 \ldots N_p$ is defined as:

$$\langle\!\langle \lambda\overline{\xi}.f N_1 \ldots N_p \rangle\!\rangle = \langle \langle\!\langle N_1 \rangle\!\rangle, \ldots, \langle\!\langle N_p \rangle\!\rangle \rangle \fatsemi^{0..p-1} [\![f]\!]$$

where $[\![f]\!]$ is the standard strategy denotation of f.

4.3.1.4 Correspondence theorem

We now show how to extend the Correspondence Theorem of the simply-typed lambda calculus (Theorem 4.96) to PCF.

Lemma 4.102. *Let $(S, \subseteq)$ denote the set of sets of justified sequences of nodes ordered by subset inclusion. The function $\mathcal{T}rav(\_)^{\upharpoonright \circledast} : (CT, \sqsubseteq) \rightarrow (S, \subseteq)$ is continuous.*

Proof. - *Monotonicity*: Let T and T' be two computation trees such that $T \sqsubseteq T'$ and let t be some traversal of T. Traversals ending with a node labelled $\perp$ are maximal therefore $\perp$ can only occur at the last position in a traversal. We prove the following properties:

(i) If $t = t \cdot n$ with $n \neq \perp$ then t is a traversal of T';

(ii) if $t = t_1 \cdot \perp$ then $t_1 \in \mathcal{T}rav(T')$.

(i) By induction on the length of t. It is trivial for the empty traversal. Suppose that $t = t_1 \cdot n$ is a traversal where $n \neq \perp$ and t_1 is a traversal of T'. We observe that in all traversal rules, the produced traversal is of the form $t_1 \cdot n$ where n is defined to be a child node or value-leaf of some node m occurring in t_1. Moreover because constant rules are well-behaved, the choice of the node n only depends on the traversal t_1.

Since $T \sqsubseteq T'$, any node m occurring in t_1 belongs to T' and the children nodes of m in T also belong to the tree T'. Hence n is also present in T' and the rule used to produce the traversal t of T can be used to produce the traversal t of T'.

(ii) $\perp$ can only occur at the last position in a traversal therefore t_1 does not end with $\perp$ and by (i) we have $t_1 \in \mathcal{T}rav(T')$.

Hence we have:

$$\begin{aligned}
\mathcal{T}rav(T)^{\upharpoonright\circledast} &= \{t \upharpoonright r \mid t \in \mathcal{T}rav(T)\} \\
&= \{(t \cdot n) \upharpoonright r \mid t \cdot n \in \mathcal{T}rav(T) \wedge n \neq \perp\} \cup \{(t \cdot \perp) \upharpoonright r \mid t \cdot \perp \in \mathcal{T}rav(T)\} \\
\text{(by (i) and (ii))} \quad &\subseteq \{(t \cdot n) \upharpoonright r \mid t \cdot n \in \mathcal{T}rav(T') \wedge n \neq \perp\} \cup \{t \upharpoonright r \mid t \in \mathcal{T}rav(T')\} \\
&= \mathcal{T}rav(T')^{\upharpoonright\circledast} \ .
\end{aligned}$$

- *Continuity*: Let $t \in \mathcal{T}rav\left(\bigcup_{n\in\omega} T_n\right)$. We write t_i for the finite prefix of t of length i. The set of traversals is prefix-closed therefore $t_i \in \mathcal{T}rav\left(\bigcup_{n\in\omega} T_n\right)$ for every i. Since t_i has finite length we have $t_i \in \mathcal{T}rav(T_{j_i})$ for some $j_i \in \omega$. Therefore we have:

$$\begin{aligned}
t \upharpoonright r &= (\bigvee_{i\in\omega} t_i) \upharpoonright r && \text{(the sequence } (t_i)_{i\in\omega} \text{ converges to } t) \\
&= \bigcup_{i\in\omega} (t_i \upharpoonright r) && \text{since } _ \upharpoonright r \text{ is continuous (Lemma 4.15)} \\
&\in \bigcup_{i\in\omega} \mathcal{T}rav(T_{j_i})^{\upharpoonright\circledast} && \text{since } t_i \in \mathcal{T}rav(T_{j_i}) \\
&\subseteq \bigcup_{i\in\omega} \mathcal{T}rav(T_i)^{\upharpoonright\circledast} && \text{since } \{j_i \mid i \in \omega\} \subseteq \omega.
\end{aligned}$$

Hence $\mathcal{T}rav(\bigcup_{n\in\omega} T_n)^{\upharpoonright\circledast} \subseteq \bigcup_{n\in\omega} \mathcal{T}rav(T_n)^{\upharpoonright\circledast}$. ∎

Proposition 4.103. *Let $\Gamma \vdash M : T$ be a PCF term and r be the root of $\tau(M)$. Then:*

$$\begin{aligned}
(i) &\quad \varphi_M(\mathcal{T}rav(M)^*) = \langle\!\langle M \rangle\!\rangle \ , \\
(ii) &\quad \varphi_M(\mathcal{T}rav(M)^{\upharpoonright\circledast}) = [\![M]\!] \ .
\end{aligned}$$

Proof. We first show the result for PCF$_1$: For (i), the proof is an induction identical to the proof of Theorem 4.96; we just need to complete it with the new constants cases. The cases `succ`, `pred`, `cond` and numeral constants are straightforward. Case $M \equiv \Omega_o$: We have $\mathcal{T}rav(\Omega_o) = \mathsf{Pref}(\{\lambda \cdot \perp\})$ therefore $\mathcal{T}rav(\Omega_o)^{\upharpoonright\circledast} = \mathsf{Pref}(\{\lambda\})$ and $[\![\Omega_o]\!] = \mathsf{Pref}(\{q\})$ with $\varphi(\lambda) = q$. Hence $[\![\Omega_o]\!] = \varphi(\mathcal{T}rav(\Omega_o)^{\upharpoonright\circledast})$. (ii) is a direct consequence of (i) and the Projection Lemma 4.89.

We now extend the result to PCF. Let M be a PCF term, we have:

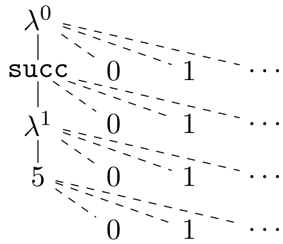
$$\begin{aligned}
[\![M]\!] &= \bigcup_{n\in\omega} [\![M_n]\!] && \text{[AM98b, lemma 16]} \\
&= \bigcup_{n\in\omega} \mathcal{T}rav(\tau(M_n))^{\upharpoonright\circledast} && \text{since } M_n \text{ is a PCF}_1 \text{ term} \\
&= \mathcal{T}rav(\bigcup_{n\in\omega} \tau(M_n))^{\upharpoonright\circledast} && \text{by continuity of } \mathcal{T}rav(_)^{\upharpoonright\circledast}\text{, Lemma 4.102}
\end{aligned}$$

$$\begin{aligned} &= \mathcal{T}rav(\tau(M))^{\upharpoonright \circledast} && \text{by definition of } \tau(M) \\ &= \mathcal{T}rav(M)^{\upharpoonright \circledast} \ . && \square \end{aligned}$$

Hence by Corollary 4.94, φ defines a bijection from $\mathcal{T}rav(M)^{\upharpoonright \circledast}$ to $[\![M]\!]$:

$$\varphi : \mathcal{T}rav(M)^{\upharpoonright \circledast} \stackrel{\cong}{\longrightarrow} [\![M]\!] \ .$$

Example 4.104 (Successor operator)**.** Consider the term $M \equiv \texttt{succ}\ 5$ whose computation tree is represented below. Vertices attached to their parent node with a dashed line represent the value-leaves.



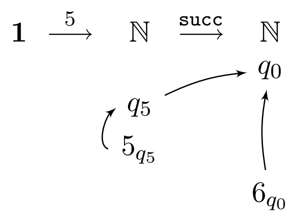
The following sequence of nodes is a traversal of $\tau(M)$:

$$t = \lambda^0 \cdot \texttt{succ} \cdot \lambda^1 \cdot 5 \cdot 5_5 \cdot 5_{\lambda^1} \cdot 6_{\texttt{succ}} \cdot 6_{\lambda^0} \ .$$

The subsequences t^* and $t \upharpoonright r$ are given by:

$$t^* = \lambda^0 \cdot \lambda^1 \cdot 5_{\lambda^1} \cdot 6_{\lambda^0} \qquad \text{and} \qquad t \upharpoonright r = \lambda^0 \cdot 6_{\lambda^0} \ .$$

The sequence $\varphi(t^*) = q_0 \cdot q_5 \cdot 5_{q_5} \cdot 5_{q_0}$, where q_0 and q_5 both denote the root of the flat arena over $\mathbb{N}$, is a play of the syntactically-revealed semantics; the sequence $\varphi(t \upharpoonright r) = q_0 \cdot 5_{q_0}$ is a play of the standard semantics. The interaction play $\varphi(t^*)$ is represented below:



Example 4.105 (Conditional)**.**

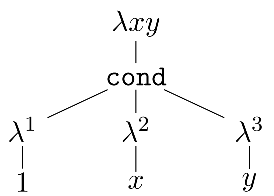


Figure 4.5: Computation tree of the term $\lambda xy.\texttt{cond}\ 1\ x\ y$.

Take the computation tree represented on the left (value-leaves are not shown). For every value $v \in \mathcal{D}$ we have the following traversal:

$$t = \lambda xy \cdot \texttt{cond} \cdot \lambda^1 \cdot 1 \cdot 1_1 \cdot \lambda^1_1 \cdot \lambda^3 \cdot y \cdot v_y \cdot v_{\lambda^3} \cdot v_{\texttt{cond}} \cdot v_{\lambda xy} \ .$$

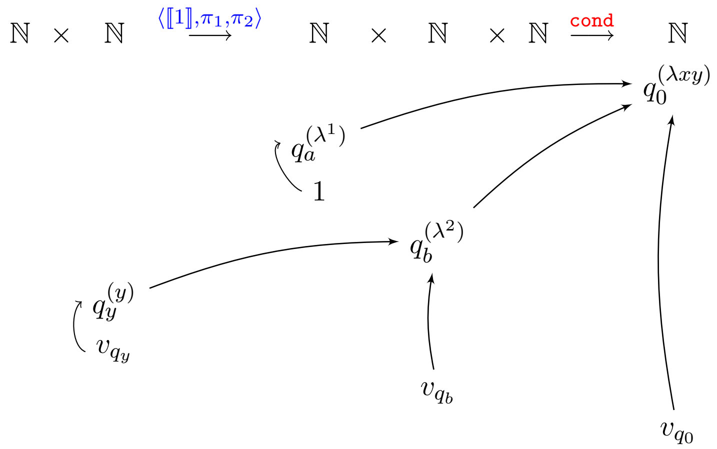
The subsequence t^* is given by:

$$t^* = \lambda xy \cdot \lambda^1 \cdot \lambda^1_1 \cdot \lambda^3 \cdot y \cdot v_y \cdot v_{\lambda^3} \cdot v_{\lambda xy}$$

and the core of $t \upharpoonright \circledast$ is given by:

$$t \upharpoonright \circledast = \lambda xy \cdot y \cdot v_y \cdot v_{\lambda xy} \ .$$

By the correspondence theorem, the sequence of moves $\varphi(t^*)$ (represented in the diagram below) is a play of the revealed semantics, and the sequence $\varphi(t \upharpoonright \circledast)$ is the play of the standard semantics obtained by hiding the internal moves from $\varphi(t^*)$.



Remark 4.106 (Finite representation of the computation tree) Due to the presence of the Y combinator, computation trees of PCF terms are potentially infinite. It is possible to give an equivalent finite representation using computation *graphs*. We briefly describe how this can be achieved.

The idea is to replace Y-recursion by μ-recursion: each subterm of the form $\mathsf{Y}_A\ M$ is replaced by $\mu f.M f$ for f fresh. The computation graph is then obtained from the eta-long normal form of the term. The abstraction nodes are generalized to take into account μ binders: an abstraction node is of the form $\lambda\!\!\lambda\overline{x}$ where $\overline{x}$ is a list of μ-bound and λ-bound variables where the μ-bound variables are written in parenthesis to distinguish them from λ-bound variables.

The computation graph of $\mathsf{Y}_A(\lambda f^A.M)$ for $A = (A_1, \ldots, A_n, o)$ is then obtained from the syntax representation of $\lambda\!\!\lambda(f)x_1 \ldots x_n.\lceil M\rceil$ by adding a child edge going from each occurrence of the recursion variable f in $\lceil M\rceil$ to the root $\lambda\!\!\lambda(f)x_1 \ldots x_n$.

This presentation also accounts for ground type recursion, for instance the computation graph of the `while` operator of Idealized Algol defined as `while` C `do` $I \equiv Y(\lambda f.$`cond` C `skip` (`seq` $I f$)) is given by the graph of $\lambda\!\!\lambda(f).$`cond` C `skip` (`seq` $I f$).

The order of a generalized abstraction node is still defined as the order of the term represented by the subtree rooted at this node. In other word, the order of $\lambda\!\!\lambda\overline{x}$ is defined as the order of $\lambda\!\!\lambda\overline{y}$ where $\overline{y}$ is the sublist of $\overline{x}$ obtained by removing all the recursion variables (those in parenthesis).

Bound variables in a generalized abstraction node $\lambda\!\!\lambda\overline{x}$ are numbered as follows: The i^{th} λ-bound variable in $\overline{x}$ is denoted by 'i' and the i^{th} recursion variable is denoted by '(i)'. The links in a justified sequence of nodes are labelled accordingly. The traversal rules can thus be kept unmodified: the recursion variables in the λ-nodes are ignored by the rules since such variables are numbered differently from standard variables. In particular, the (Var) rule only applies to non-recursion variables. We only need to add a rule to handle recursion variable: whenever a traversal meets a recursion variable f in the subgraph $\tau(F)$, the traversal jumps to the root of the graph:

> ($\mathsf{Var}_{\mathsf{rec}}$) If $t' \cdot n \cdot \lambda\!\!\lambda\overline{x} \ldots \overset{(i)}{f_i}$ is a traversal for some *recursion* variable f_i bound by $\lambda\!\!\lambda\overline{x}$ then so is $t' \cdot n \cdot \lambda\!\!\lambda\overline{x} \ldots \overset{(i)}{f_i} \cdot \lambda\!\!\lambda\overline{x}$.

The enabling relation $\vdash$ needs to be adapted to allow the root to be justified by a recursion variable (as if it was a child of the recursion variable). Since a traversal can now visit the root multiple times, the definition of the traversal core also needs to be adapted: instead of keeping all the nodes hereditarily *enabled* by the root, it keeps the nodes that are hereditarily *justified* by *some occurrence* of the root that has no justifier. The definition of the mapping ψ from nodes to moves remains consistent with this notion of computation tree, and the game-semantic correspondence follows.

4.3.2 Idealized algol

We now consider the language Idealized Algol. The general idea is the same as for PCF, however there are some difficulties caused by the presence of the two base types `var` and `com`. We briefly sketch how the framework can be adapted to IA without detailing the proof of the Correspondence Theorem.

Computation hypertree

The two languages that we have considered up to now (lambda calculus and PCF) do not have product types. Consequently, the arenas involved in their game model have a single initial move at most, and can therefore be regarded as trees. This property permitted us to represent the game denotation of term directly on some representation of its abstract syntax tree—the computation tree. This cannot be done in IA because the base type `var` is given by the product $\texttt{com}^\omega \times \texttt{exp}$ which corresponding game has infinitely many initial moves, whereas the AST of the term is a tree and therefore has a single root.

The overcome this mismatch, we use hypertrees instead of trees. These hypertrees provide an abstract representation of the syntax of the term in which some nodes, called *generalized lambda nodes*, are themselves constituted of (possibly infinitely many) subnodes. Furthermore each individual subnode can have its own children nodes.

Notations 4.107 For every type μ, we write $\mathcal{D}_\mu$ to denote the set of values of type μ. We have $\mathcal{D}_{\texttt{exp}} = \mathbb{N}$, $\mathcal{D}_{\texttt{com}} = \{\texttt{done}\}$ and $\mathcal{D}_{\texttt{var}} = \mathcal{D}_{\texttt{exp}} \cup \mathcal{D}_{\texttt{com}}$. For every node n, if the tree rooted at n (*i.e.*, $M^{(n)}$) is of type $(A_1, \ldots A_n, B)$ then we call B the *return type of* n. The set of value-leaves of a node n is given by $\mathcal{D}_\mu$ where μ is the return type of n. For conciseness, when representing value-leaves in the hypertree, we merge all the value-leaves of a given node of type μ into a single leaf labelled $\mathcal{D}_\mu$. For instance we use the tree notation

$$\begin{array}{c} n \\ | \\ \mathcal{D}_{\texttt{exp}} \end{array} \quad \text{to mean} \quad \begin{array}{c} n \\ \swarrow\ \ \swarrow\ \ \searrow\ \ \searrow \\ 0 \quad 1 \quad 2 \quad \cdots \end{array} \qquad \text{and} \quad \begin{array}{c} n \\ | \\ \mathcal{D}_{\texttt{com}} \end{array} \quad \text{for} \quad \begin{array}{c} n \\ | \\ \texttt{done} \end{array} .$$

The computation hypertree of a term with return type `var` has infinitely many root lambda-nodes which are merged all-together into a single node called a ***generalized lambda-node***. The subnodes of a generalized lambda nodes are labelled λ^r, λ^{w_0}, λ^{w_1}, λ^{w_2}, … Suppose that M is a term of type `var`, then the computation hypertree for $\lambda\overline{\xi}.M$ is obtained by relabelling the root λ-nodes λ^r, λ^{w_0}, λ^{w_1}, λ^{w_2}, … into $\lambda^r\overline{\xi}$, $\lambda^{w_0}\overline{\xi}$, $\lambda^{w_1}\overline{\xi}$, $\lambda^{w_2}\overline{\xi}$, …. If M is of type `exp` or `com` then the computation hypertree for $\lambda\overline{\xi}.M$ is computed the same way as for computation trees of lambda-terms.

Table 4.4 defines the computation hypertree for each term-construct of IA. A generalized lambda node is represented by a frame surrounding its subnodes (3^{rd} and 8^{th} row in the table).

Enabling relation, justified sequence

The notion of binder is redefined as follows: Given a variable node x, the binder of x is the first node occurring in the path to the root that is a lambda node $\lambda\overline{x}$ with $x \in \overline{x}$ or a block-declaration node $\texttt{new}\ x$. The enabling relation and the definition of justified sequence is modified so that occurrences of block-allocated variables are justified by nodes of type $\texttt{new}\ x$ instead of lambda nodes.

Children numbering convention

Let p be a node and suppose that its i^{th} child n has return type `var`. Then n is a generalized lambda node with subnodes $\lambda^r\overline{\xi}$, $\lambda^{w_0}\overline{\xi}$, …. From the point of view of the parent node p, these subnodes are referenced as "$i.\alpha$" where $0 \leq i \leq arity(p)$ and $\alpha \in \{r\} \cup \{w_k \mid k \in \mathbb{N}\}$. For

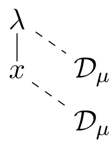
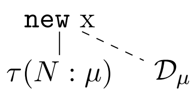
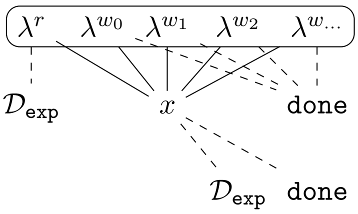
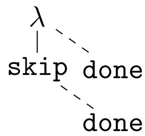
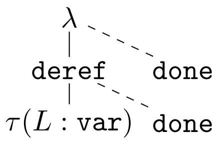
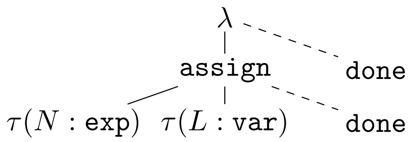
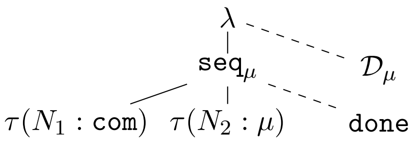
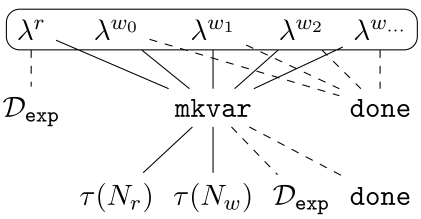
M	$\tau(M)$
$\mathtt{x} : \mu$ $\mu \in \{\mathtt{com}, \mathtt{exp}\}$	
$\mathtt{new}\ x\ \mathtt{in}\ N : \mu$ $\mu \in \{\mathtt{com}, \mathtt{exp}\}$	
$\mathtt{x} : \mathtt{var}$	
$\mathtt{skip}\!: \mathtt{com}$	
$\mathtt{deref}\ L : \mathtt{exp}$	
$\mathtt{assign}\ L\ N : \mathtt{com}$	
$\mathtt{seq}_\mu\ N_1\ N_2 : \mathtt{com}$ $\mu \in \{\mathtt{exp}, \mathtt{com}\}$	
$\mathtt{mkvar}\ N_w\ N_r : \mathtt{var}$	

Table 4.4: Computation hypertrees of IA constructs.

instance $i.r$ refers to the root labelled $\lambda^r\overline{\xi}$ of the i^{th} child of p, and $i.w_k$ refers to the root labelled $\lambda^{w_k}\overline{\xi}$.

Traversals

The following new rules are added on top of those defined in Sec. 4.1.3:

- *Application rules*

  The rule (app) is now split up in three rules ($\mathsf{app}_{\mathtt{exp}}$), ($\mathsf{app}_{\mathtt{com}}$) and ($\mathsf{app}_{\mathtt{var}}$) corresponding to traversals ending with an @-node of return type `exp`, `com` and `var` respectively. The rules ($\mathsf{app}_{\mathtt{exp}}$), ($\mathsf{app}_{\mathtt{com}}$) are defined identically to (app) (see Sec. 4.1.3). The rule ($\mathsf{app}_{\mathtt{var}}$) is

$$(\mathsf{app}_{\mathtt{var}})\ t\cdot\lambda^k\overline{\xi}\cdot @ \in \mathcal{T}rav \text{ and } k\in\{r,w_0,w_1,\ldots\} \implies t\cdot\lambda^k\overline{\xi}\cdot @\cdot\lambda^k\overline{\eta}\in\mathcal{T}rav\ .$$

- *Input-variable rules*

  We define the rules ($\mathsf{InputVal}^{\$}$) for \$ ranging in $\{\mathtt{com},\mathtt{var},\mathtt{exp}\}$. For `com` and `exp`, the rules are defined identically to ($\mathsf{InputVal}$) of Sec. 4.1.3. The `var` case is implemented by two rules:

$$(\mathsf{InputValue}^{\mathtt{var}}_{\mathsf{r}})\ \frac{t_1\cdot\lambda^r\overline{\xi}\cdot x\cdot t_2\in\mathcal{T}rav}{t_1\cdot x\cdot t_2\cdot v_x\in\mathcal{T}rav}\ x \text{ pending node } \wedge\ x\in IN^{\circledast\vdash}_{\mathsf{var}}\wedge x:\mathtt{var},\ v\in\mathcal{D}\ .$$

$$(\mathsf{InputValue}^{\mathtt{var}}_{\mathsf{w}})\ \frac{t_1\cdot\lambda^w\overline{\xi}\cdot x\cdot t_2\in\mathcal{T}rav}{t_1\cdot x\cdot t_2\cdot\mathtt{done}_x\in\mathcal{T}rav}\ x \text{ pending node } \wedge\ x\in IN^{\circledast\vdash}_{\mathsf{var}}\wedge x:\mathtt{var}\ .$$

- *IA constants rules*

  The rules for the constants of IA are given in Table 4.5. These rules for `new` are purely structural, they are defined similarly to ($\mathsf{app}_{\mathtt{exp}}$), ($\mathsf{app}_{\mathtt{com}}$) and ($\mathsf{app}_{\mathtt{done}}$).

  The rules from Table 4.5 do not suffice to model `mkvar` however. We need to specify what happens when reaching a variable node that is hereditarily justified by the constant `mkvar`. Take for instance the term `assign` (`mkvar` $(\lambda x.M)N)7$. The rule (mkvar$''_w$) permits one to pass the node `mkvar` and to continue with the traversal of the computation tree of $\lambda x.M$, which may subsequently lead to some occurrence of x. The behaviour of the traversal at this point is specified by the rules defined in the next paragraph.

- *Variable rules*

  Let x be an internal variable node. Then by definition it is either hereditarily justified by an @-node or by a Σ-constant node.

  - Suppose that x's binder is a lambda node $\lambda\overline{x}$ and $x\in IN^{@\vdash}$.

    This case is a generalization of the rule (Var) (Sec. 4.1.3). The only difference is that for variables of type `var`, the lambda nodes preceding x in the traversal determines the lambda node that is visited next:

$$(\mathsf{Var}_{\mathtt{var}})\frac{t\cdot n\cdot\lambda\overline{x}\ldots\lambda^\alpha x_i\cdot x_i\in\mathcal{T}rav}{t\cdot n\cdot\lambda\overline{x}\ldots\lambda^\alpha x_i\cdot x_i\cdot\lambda\overline{\eta_i}\in\mathcal{T}rav}\quad x_i\in IN^{@\vdash}_{\mathsf{var}}\wedge\alpha\in\{r\}\cup\{w_i\mid i\in\mathbb{N}\}\ .$$

$$(\mathsf{deref})\,\frac{t \cdot \mathtt{deref} \in \mathcal{T}rav}{t \cdot \mathtt{deref} \cdot n \in \mathcal{T}rav} \qquad (\mathsf{deref}')\,\frac{t \cdot \mathtt{deref} \cdot n \cdot t_2 \cdot v_n \in \mathcal{T}rav}{t \cdot \mathtt{deref} \cdot n \cdot t_2 \cdot v_n \cdot v_{\mathtt{deref}} \in \mathcal{T}rav}$$

$$(\mathsf{assign})\,\frac{t \cdot \mathtt{assign} \in \mathcal{T}rav}{t \cdot \mathtt{assign} \cdot \lambda \in \mathcal{T}rav} \qquad (\mathsf{assign}')\,\frac{t \cdot \mathtt{assign} \cdot \lambda \cdot t_2 \cdot v_\lambda \in \mathcal{T}rav}{t \cdot \mathtt{assign} \cdot \lambda \cdot t_2 \cdot v_\lambda \cdot \lambda\overline{\eta} \in \mathcal{T}rav}$$

$$(\mathsf{assign}'')\,\frac{t \cdot \mathtt{assign} \cdot t_2 \cdot \lambda\overline{\eta} \cdot t_3 \cdot \mathsf{done}_{\lambda\overline{\eta}} \in \mathcal{T}rav}{t \cdot \mathtt{assign} \cdot t_2 \cdot \lambda\overline{\eta} \cdot t_3 \cdot \mathsf{done}_{\lambda\overline{\eta}} \cdot \mathsf{done}_{\mathtt{assign}} \in \mathcal{T}rav}$$

$$(\mathsf{seq})\,\frac{t \cdot \mathtt{seq} \in \mathcal{T}rav}{t \cdot \mathtt{seq} \cdot n \in \mathcal{T}rav} \qquad (\mathsf{seq}')\,\frac{t \cdot \mathtt{seq} \cdot n \cdot t_2 \cdot v_n \in \mathcal{T}rav}{t \cdot \mathtt{seq} \cdot n \cdot t_2 \cdot v_n \cdot m \in \mathcal{T}rav}$$

$$(\mathsf{seq}'')\,\frac{t \cdot \mathtt{seq} \cdot t_2 \cdot m \cdot t_3 \cdot v_m \in \mathcal{T}rav}{t \cdot \mathtt{seq} \cdot t_2 \cdot m \cdot t_3 \cdot v_m \cdot v_{\mathtt{seq}} \in \mathcal{T}rav}$$

$$(\mathsf{mkvar}_\mathsf{r})\,\frac{t \cdot \lambda^r\overline{\xi} \cdot \mathtt{mkvar} \in \mathcal{T}rav}{t \cdot \lambda^r\overline{\xi} \cdot \mathtt{mkvar} \cdot \lambda \in \mathcal{T}rav} \qquad (\mathsf{mkvar}'_\mathsf{r})\,\frac{t \cdot \mathtt{mkvar} \cdot \lambda \cdot t_2 \cdot v_\lambda \in \mathcal{T}rav}{t \cdot \mathtt{mkvar} \cdot \lambda \cdot t_2 \cdot v_\lambda \cdot v_{\mathtt{mkvar}} \in \mathcal{T}rav}$$

$$(\mathsf{mkvar}_\mathsf{w})\,\frac{t \cdot \lambda^{w_k}\overline{\xi} \cdot \mathtt{mkvar} \in \mathcal{T}rav}{t \cdot \lambda^{w_k}\overline{\xi} \cdot \mathtt{mkvar} \cdot \lambda\overline{\eta} \in \mathcal{T}rav}$$

$$(\mathsf{mkvar}''_\mathsf{w})\,\frac{t \cdot \lambda^{w_k}\overline{\xi} \cdot \mathtt{mkvar} \cdot \lambda\overline{\eta} \cdot t_2 \cdot \mathsf{done}_{\lambda\overline{\eta}} \in \mathcal{T}rav}{t \cdot \lambda^{w_k}\overline{\xi} \cdot \mathtt{mkvar} \cdot \lambda\overline{\eta} \cdot t_2 \cdot \mathsf{done}_{\lambda\overline{\eta}} \cdot \mathsf{done}_{\mathtt{mkvar}} \in \mathcal{T}rav}$$

where v denotes some value from $\mathcal{D}$.

Table 4.5: Traversal rules for IA constants.

– Suppose that x's binder is a lambda node and $x \in IN^{IN_{\Sigma}\vdash}$. Then x's binder is necessarily the second child of a mkvar-node (since mkvar is the only constant of order greater than 0).

$$(\texttt{mkvar-Var})\frac{t \cdot \lambda^{w_k}\overline{\xi} \cdot \texttt{mkvar} \cdot \lambda x \cdot t_2 \cdot x \in \mathcal{T}rav}{t \cdot \lambda^{w_k}\overline{\xi} \cdot \texttt{mkvar} \cdot \lambda x \cdot t_2 \cdot x \cdot k_x \in \mathcal{T}rav} .$$

– Suppose that x is a block-allocated variable.

Given a block-declaration new x, we call *assignment of* x any segment of traversal of the form $\lambda^{w_k}\overline{\xi} \cdot x$ for some $k \in \mathcal{D}_{\texttt{exp}}$ and occurrence x of a node bound by new x. We call k the *value assigned* to x.

$$(\texttt{new-Var}_{\mathsf{w}}) \frac{t \cdot \lambda^{w_k}\overline{\xi} \cdot x \in \mathcal{T}rav}{t \cdot \lambda^{w_k}\overline{\xi} \cdot x \cdot \texttt{done}_x \in \mathcal{T}rav} \; x \in IN^{\texttt{new}\vdash}_{\mathsf{var}} .$$

$$(\texttt{new-Var}_{\mathsf{r}}) \frac{t_1 \cdot \texttt{new}\ x \cdot t_2 \cdot \lambda^r \overline{\xi} \cdot x \in \mathcal{T}rav}{t_1 \cdot \texttt{new}\ x \cdot t_2 \cdot \lambda^r \overline{\xi} \cdot x \cdot k_x \in \mathcal{T}rav}$$

where $k \in \mathbb{N}$ is the last value assigned to x in t_2, or 0 if there is no such assignment.

4.3.2.1 Game semantics correspondence

The properties that we proved for computation trees and traversals of the lambda calculus with constants can easily be lifted to computation hypertrees of IA. In particular:

- Constant traversal rules are well-behaved (for order-0 and order-1 constants, this is a consequence of Lemma 4.28; for mkvar and new this can be easily verified);
- P-view of traversals are paths in the computation hypertrees;
- For beta-normal terms, the P-view of a traversal core is the core of the P-view (Lemma 4.66, and the O-view of a traversal is the O-view of its core (Lemma 4.64);
- There is a mapping from nodes of the computation hypertrees to moves in the revealed game semantics;
- There is a correspondence between traversals of the computation tree and plays in revealed game semantics;
- Consequently, there is a correspondence between the standard game semantics and the set of justified sequences of nodes $\mathcal{T}rav(M)^{\upharpoonright \circledast}$.

4.4 Conclusion and related works

We have given a new presentation of game semantics based on the theory of traversals. This presentation is concrete in the sense that the traversal denotation carries syntactic information about the term. We established the connection with the Hyland-Ong game semantics by means of a Correspondence Theorem: The set of traversals of a term is isomorphic to the revealed game denotation of the term.

One advantage of the traversal theory lies in its ability to compute beta-reduction locally without having to perform term substitution. As observed by Danos et al. [DHR96], "the interaction processes at work in game semantics are implementations of *linear head reduction*". In that regards, the traversals theory can be viewed as an implementation of the *head linear reduction strategy* [DR04]. Although the idea of evaluating a term using this strategy is not new, our presentation has several advantages and novelties. Firstly, the Correspondence Theorem establishes a clear correspondence with game semantics, namely that traversals gives you a way to compute precisely the revealed game denotation of a term. To our knowledge, although the notion of revealed game semantics was mentioned in previous works [Gre04], it was never formally defined. Secondly, our presentation highlights more clearly the algorithmic aspect of game semantics. The inductive definition of traversals lends itself well to automaton characterization. An example is the characterization of higher-order recursion schemes by *collapsible higher-order pushdown automata* [HMOS08].

Another advantage of the traversal theory is its efficiency for effectively computing the game-semantic denotation of a term. The traditional approach is to proceed bottom-up by appealing to compositionality. Although the compositional nature of game semantics is very attractive from a theoretical point of view, in practice it is not efficient to compute a denotation in that way. Indeed, for every subterm one has to compute all the possible ways to interact with the environment for that subterm. But this denotation is then immediately composed with another subterm, which determines part of the environment's behaviour, thus it was wasteful in the first place to consider all the possible behaviours of the environment for the first term.

The traversal theory follows a top-down approach which means that we only consider possible behaviour of the outermost environment. Moreover contrary to the compositional method, there is no need to implement any composition mechanism: the set of traversals is just obtained by following the traversal rules; the hiding of internal nodes is postponed until the end.

The lazy nature of the traversal evaluation provides a further source of efficiency: the beta-redexes are computed "on-demand" instead of performing a global substitution.

Last but not least, we believe that the syntactic correspondence between game semantics and its syntax is of pedagogical interest. Game semantics is often found hard to understand due to some obscure technical definitions. A concrete presentation such as the one given by the traversal theory, allows one to explain game-semantic concepts (such as P-view, innocence, visibility) from a programmer point of view. I have implemented a prototype tool using the F# programming language, which among other things, illustrates the theory of traversals [Blu08]. The tool lets the user "play" the game induced by a simply-typed term (or a higher-order grammar) just by choosing nodes from the computation tree. As the game unfolds the corresponding traversal is shown. A calculator mode allows the user to perform various operations on justified sequences. (All the examples from this chapter were generated using this tool.)

Further correspondences

The traversal theory that we have presented here captures the lambda calculus fragment of the game model of call-by-name programming languages such as PCF and Idealized Algol. A natural way to extend this work would be to define the appropriate notion of traversal corresponding to the call-by-value games [Plo75, AM98a].

Applications

The theory of traversal has applications in several domains of research:

Verification

The theory of traversal was originally introduced by Ong to study the decidability of MSO theories of infinite trees generated by higher-order recursion schemes. This result was recently used by Kobayashi to develop a novel framework for verification of temporal properties of higher-order functional programs [Kob09].

Another promising application of the traversal theory concerns the study of reachability problems. In its most general form, the reachability problem for programming languages can informally be stated as: *Given a term M and coloured subterm N, is there a context $C[-]$ such that evaluating $C[M]$ involves the evaluation of N?*. In an ongoing research project, Luke

Ong and Nikos Tzevelekos make use of the traversal theory to study several variations of the reachability problem for finitary PCF [OT].

Automata theory

The traversal theory has led to an equi-expressivity result between a certain type of automaton device called *collapsible pushdown automaton* (CPDA) and higher-order recursion schemes (HORS) [HMOS08]. One direction of this proof relies on the traversal theory: for a given HORS, a CPDA is constructed that computes precisely the set of traversals over the computation tree of the HORS.

A crucial point in this encoding is that structures generated by recursion schemes are of ground type. Because such structures do not interact with the environment, their game-semantic denotation is relatively simple. In particular, the O-view of the traversal does not play any role in the traversal rules and therefore the automaton does not need to calculate or remember it. A natural extension would be a similar automata-characterization for *higher-order* structures such as simply-typed terms.

Pattern matching

Higher-order matching is the following problem: *Given an equation $M = N$ where M is an open simply-typed term and N is a closed simply-typed term, is there a solution substitution θ such that $M\theta$ and N have the same $\beta\eta$-normal form?* Huet conjectured in 1976 that this problem is decidable [Hue76]. It was proved only recently by Colin Stirling that it is indeed the case [Sti06].

Stirling's argument is based on a game-theoretic argument, namely the concept of tree-checking games. As pointed out by Luke Ong, Stirling's games are closely related to the innocent game semantics framework provided by the theory of traversals. The concept of traversals is implicitly present in Stirling's proof (though the notion of justification pointers is replaced by "iteratively defined look-up tables").

Analyzing syntactic constraints

The connection between syntax and semantics provided by the traversal theory enables us to analyze the effect of a given syntactic constraint on the game model. The next chapter is an example of such an application: By making simple observations about the computation tree of safe terms, the Correspondence Theorem allows us to show that their strategy denotations are of a particular kind: Their plays satisfy a certain property called *incremental justification*.

Chapter 5

Syntactic Analysis of the Game Denotation of Safe Terms

Our aim is to characterize safety by game semantics. This chapter assumes that the reader is familiar with the basics of game semantics introduced in Chapter 2. Recall that a *justified sequence* over an arena is an alternating sequence of O-moves and P-moves such that every move m, except the opening move, has a pointer to some earlier occurrence of the move m_0 such that m_0 enables m in the arena. A *play* is just a justified sequence that satisfies Visibility and Well-Bracketing. A basic result in game semantics is that lambda-terms are denoted by *innocent strategies*, which are strategies that depend only on the *P-view* of a play.

In this chapter we give a precise game-semantic characterization of the safety restriction (Theorem 5.19): if a lambda-term is safe, then its game semantics (is an innocent strategy that) is, what we call, *P-incrementally justified*. In such a strategy, pointers emanating from the P-moves of a play are uniquely reconstructible from the underlying sequence of moves and pointers from the O-moves therein: specifically a P-question always points to the last pending O-question (in the P-view) of a greater order. A consequence of this characterization is that that pointers in a play of a strategy denoting a safe term can be uniquely recovered from O-questions' justification pointers and from the underlying sequence of moves (Proposition 5.21).

Since the safety condition is a syntactic constraint, and because game models are in essence syntax-independent, giving a game semantic characterization is not obvious. The Correspondence Theorem from Chapter 4 provides a way to reason syntactically about the game denotation of a term: it relates the strategy denotation of a lambda-term M to the set of *traversals* over a souped-up abstract syntax tree of the η-long normal form of M. More precisely, in the language of game semantics, it says that traversals are just (concrete representations of) the *uncovering* (in the sense of Hyland and Ong [HO00]) of plays in the strategy denotation. Based on this result, showing Theorem 5.19 just amounts to making some simple observations on the syntax of safe terms.

In the first section we introduce the notion of *P-incrementally justified strategies*. In the following two sections we introduce the notion of *incrementally-bound computation trees* and establish a relationship between incremental-binding and P-incremental justification (Proposition 5.14). Finally, we show that safe simply-typed terms have incrementally-bound computation trees, consequently their game denotation is P-incrementally justified.

The fourth section concerns the game-semantic characterization of the safe lambda calculus without interpreted constants. We then extend the result by taking into account the interpreted constants of PCF and IA: we show that safe PCF and safe IA terms are denoted by P-incrementally justified strategies.

Some of the results presented in this chapter were first published in TLCA [BO07]. They are reproduced here with complete proofs and generalized to the languages PCF and IA.

5.1 P-incrementally justified strategies

In the game semantics literature, some authors use the term "order of a question move" to refer to the length of the path in the arena to the initial move that hereditarily enables it. For the purpose of studying the safety restriction it will be convenient instead to call it the ***level*** of the node, and reserve the term "order" to refer to another quantity: The ***order of a question move*** q, written $\operatorname{ord} q$, is defined as the length of the longest enabling-chain of questions starting from q minus 1 (see Def 2.74). Thus the order of an arena can be defined in term of move-order: it is precisely the greatest order of its initial moves.

Definition 5.1. A play sm of even length is said to be ***P-incrementally justified***, or *P-i.j.* for short, if m points to the last unanswered O-question in $\ulcorner s \urcorner$ with order strictly greater than $\operatorname{ord} m$. A strategy σ is said to be ***P-incrementally justified***, if all plays in σ ending with a P-question are P-incrementally justified.

Note that although a P-question's *pointer* is determined by the P-view, the choice of the question move itself can be based on the whole history of the play. Thus P-incremental justification does not imply innocence.

The definition suggests an algorithm that, given a play of a P-incrementally justified denotation, uniquely recovers the pointers from the underlying sequence of moves and from the pointers associated to the O-moves therein. Hence:

Lemma 5.2. *In P-incrementally justified strategies, pointers emanating from P-moves are superfluous.*

Proof. Suppose σ is a P-incrementally justified strategy. We prove that pointers attached to P-moves in a play $s \in \sigma$ are uniquely recoverable by induction on the length of s. *Base case*: If $|s| \leq 1$ then there is no pointer to recover. *Step case*: Suppose $sm \in \sigma$. If m is an answer move then by the well-bracketing condition m points to the last unanswered question in s. If m is a P-question then by P-incremental justification of σ, m points to the last O-move in $\ulcorner s \urcorner$ with order strictly greater than $\operatorname{ord} q$. Since we have access to O-moves' pointers, we can compute the P-view $\ulcorner s \urcorner$. Hence m's pointer is uniquely recoverable. $\square$

Example 5.3. Copycat strategies, such as the identity strategy id_A on game A or the evaluation map $ev_{A,B}$ of type $(A \rightarrow B) \times A \rightarrow B$, are all P-incrementally justified.[1]

5.2 Dead code elimination

We recall that the β-normal form of a term of an applied lambda calculus is the (possibly infinite) term obtained by reducing all the β-redexes. Because of the presence of interpreted constants, a β-normal form is not necessarily normal with respect to the small-step semantics. For instance in PCF, the term $\mathtt{cond}\,0\,M\,N$ is β-normal but it reduces in one step to M.

We say that a coloured subterm N of $M : (A_1, \ldots, A_n, o)$ is ***dead code*** if for every context $C[-]$ such that $C[M]$ is of ground type, every reduction sequence starting from $C[M]$ does not involve a reduction of the subterm N; formally, there is no reduction sequence of the form $C[M] \rightarrow \ldots \rightarrow E[\sigma(N)] \rightarrow E[N']$ for some evaluation context $E[-]$, term N', and substitution σ of free variables of N.

Example 5.4. The subterm N in $\mathtt{cond}\,0\,M\,N$ is dead-code, whereas in $\lambda x.(\mathtt{cond}\,0\,x\,N)\,M$ the subterm x is not dead-code.

[1] In such strategies, a P-move m is justified as follows: either m points to the immediately preceding move in the P-view, or the preceding move is of smaller order and m is justified by the second last O-move in the P-view.

The dead code elimination problem is the converse of the ***reachability problem***: Given a term M containing a coloured subterm N of M, is there a context $C[-]$ such that $C[M]$ is of ground type, and a reduction sequence $C[M] \rightarrow \ldots \rightarrow E[\sigma(N)]$ for some evaluation context $E[-]$ and substitution σ of free variables of N? The reachability problem is clearly not trivial. In fact for PCF it is not decidable since the halting problem for PCF—which is a Turing-complete language—can be encoded into a reachability problem.

Let M be a term in eta-long normal form. Occurrences of variables that are in the dead code of M are called ***dead occurrences***. Given a term M, we define M^* as the term obtained from the (possibly infinite) η-long normal form of M by substituting all subterms of the form $xN_1 \ldots N_k$ where $x : (B_1, \ldots, B_k, o)$ is a dead variable occurrence by the constant $\perp$ of type o. This process is called ***dead variable elimination***. We write $\tau(M)^*$ to denote the equivalent transformation on the computation tree of M.

Clearly we have:

$$\mathcal{T}rav(M^*) \subseteq \mathcal{T}rav(M) \ . \tag{5.1}$$

Reachability by traversals

A node of a computation tree is said to be ***reachable*** if there exists a traversal that visits it. By the Correspondence Theorem, it is easy to show that if a node is not *reachable* then the corresponding variable occurrence is a dead occurrence:

Lemma 5.5. *If x is a variable node in $\tau(M)^*$ then the corresponding node in $\tau(M)$ is reachable by some traversal.*

Proof. By a game-semantic argument: suppose that there is a reduction sequence leading to the evaluation of a variable x then by soundness of the game model there must be a play of the revealed denotation $\langle\langle M \rangle\rangle$ containing the corresponding question-move. Hence by the Correspondence Theorem there must be a traversal of $\tau(M)$ that reaches the corresponding node in the computation tree. $\square$

However the converse does not hold. This is because the Correspondence Theorem concerns the *intensional* innocent game model where the Opponent is not restricted to play deterministically, let alone innocently. In this model, the strategy denotation accounts for contexts $C[-]$ that are not part of the language considered, whereas in the definition of dead-code elimination, the context ranges over terms of the language only. Hence a variable node may be reachable by a traversal (as defined in Chapter 4) but not reachable in the sense defined above (with respect to the operational semantics of the language).

Example 5.6. Take the following simply-typed lambda-term:

$$M \equiv \lambda \varphi^{(o,o)}\, x^o\, y^o\, z^o . \varphi x (\varphi y z) \ .$$

The node of its computation tree corresponding to the occurrence y is reachable by the traversal $\lambda\varphi xyz \cdot \varphi \cdot \lambda \cdot \varphi \cdot \lambda \cdot y$ but there is no simply-typed context $C[-]$ such that the evaluation of $C[M]$ leads to the evaluation of y.

Remark 5.7 In order to reconcile the two notions of reachability, one needs to enforce O-innocence in the rules of Table 4.3 so that whenever a lambda node is visited, it is uniquely determined by the O-view of the traversal at that point. This requires one to impose a global condition on the set of traversals generated by the rules ensuring consistency with respect to O-innocence (*i.e.*, for any two generated traversals $t_1 \cdot n_1$, $t_2 \cdot n_2$, if $\llcorner t_1 \lrcorner = \llcorner t_2 \lrcorner$ then n_1 and n_2 are the same node and points to the same occurrence in the O-view).

5.3 Incremental binding

In this section, we work in the general setting of an applied simply-typed lambda calculus extended with a stock of interpreted constants Σ (but without recursion), whose terms-in-context are of the form $\Gamma \vdash M : T$. We consider its safe fragment, as defined in Sec 3.5.3, whose terms-in-context are written $\Gamma \vdash_{\mathsf{s}} M : T$.

We assume that a game-semantic model is defined for this unspecified language and we write $[\![\Gamma \vdash M : T]\!]$ to denote the strategy denotation of $\Gamma \vdash M : T$. We further assume that there are *well-behaved* (see Def. 4.27) traversal rules modeling the behaviour of the constants in such a way that the game-semantic correspondence (Theorem 4.96) holds for that language. We now fix a term-in-context $\Gamma \vdash M : T$ for the rest of this section.

NOTATIONS 5.8 We call ***path*** any sequence of nodes such that for every two consecutive nodes $a \cdot b$ in the sequence, a is the parent of b. We write $[n_1, n_2]$ to denote, if it exists, the unique path going from node n_1 to node n_2 equipped with the justification pointers induced by the enabling relation $\vdash$ (A node has a unique enabler in the path to the root thus for each occurrence in $[n_1, n_2]$ there is at most one occurrence of its enabler in $[n_1, n_2]$). We write $]n_1, n_2]$ for the sub-sequence of $[n_1, n_2]$ obtained by removing n_1 together with all the associated pointers.

The symbol $\circledast$ denotes the root of the computation tree $\tau(M)$, $IN^{\circledast\vdash}$ denotes the subset of N consisting of nodes that are hereditarily enabled by $\circledast$, and $IN^{\Sigma\vdash}$ denotes the nodes that are hereditarily enabled by some constant in Σ.

Definition

Recall from the definition of computation trees (Chapter 4) that a variable node n labelled x is said to be *bound* by a node m if m is the closest node in the path from n to the root such that m is labelled $\lambda\overline{\xi}$ with $x \in \overline{\xi}$. Thus the binder node always occurs in the path from the variable node that it binds to the root. We now introduce a class of computation trees in which the binder node is uniquely determined by the nodes' orders.

Definition 5.9 (Incrementally-bound computation tree)**.** Let A be a subset of nodes of the computation tree.

(i) A variable node x of a computation tree is said to be ***A-incrementally-bound*** if its enabler is the first λ-node from A in the path to the root that has order strictly greater than $\operatorname{ord} x$. Formally:

$$x \text{ is } A\text{-incrementally-bound} \iff \begin{cases} x \text{ is enabled by } b \in [\circledast, x] \cap A \ ; \\ \operatorname{ord} b > \operatorname{ord} x \\ \forall \lambda\text{-node } n' \in]n, x] \cap A. \operatorname{ord} n' \leq \operatorname{ord} x \ . \end{cases}$$

This definition can be split into two cases:

(a) x is *bound* by the first λ-node from A occurring in the path to the root that has order strictly greater than $\operatorname{ord} x$.

(b) or x is a *free variable* and all the λ-nodes from A occurring in the path to the root except the root have order smaller or equal to $\operatorname{ord} x$.

(ii) A computation tree is said to be ***A-incrementally-bound***, also abbreviated A-i.b., if all the variable nodes from A are A-incrementally-bound

(iii) A node (resp. a tree) is ***incrementally-bound*** if it is $(N \setminus IN^{\Sigma\vdash})$***-incrementally-bound*** where N is the entire set of nodes of the computation tree and $IN^{\Sigma\vdash}$ is the set of nodes hereditarily justified by some constant node.

Lemma 5.10.

(i) For every two sets of nodes A and B satisfying $A \subseteq B$, B-incremental-binding implies A-incremental-binding.

(ii) $\tau(M)$ is A-incrementally bound if and only if $\tau(\mathsf{closure}(M))$ is.

where $\mathsf{closure}(M)$ denotes the closed term obtained by abstracting the free variables in M (see Sec. 2.1).

Proof. (i) follows immediately from the definition. (ii) This is because the computation trees $\tau(M)$ and $\tau(\mathsf{closure}(M))$ are isomorphic and the enabling relation $\vdash$ is defined identically on these two trees (since free variable nodes are enabled by the root). $\square$

Safety and incremental binding

We recall that a term is *almost safe* if it can be written $\lambda x_1 \ldots x_n.N_0 \ldots N_p$ for some $n, p \geq 0$ where N_i is safe for all $0 \leq i \leq p$. It is an *almost safe application* if further $n = 0$ (*i.e.*, no abstraction).

Proposition 5.11 (Safe terms have incrementally-bound computation trees)**.** *Let $\Gamma \vdash M : T$ be a term of some applied typed lambda calculus (without recursion).*

(i) If M is almost safe then $\tau(M)$ is incrementally-bound ;

(ii) conversely, if $\tau(M)$ is incrementally-bound then the η-long normal form of M is almost safe, and safe if further M is closed.

Proof. (i) Suppose that M is almost safe. Computation trees are defined modulo eta-long normal expansion thus since this transformation preserves almost safety (Lemma 3.41) we can assume that M is in eta-long nf. By the previous lemma, to show that $\tau(M)$ is incrementally-bound we just have to show that $\tau(\mathsf{closure}(M))$ is incrementally-bound. We now consider $\tau(\mathsf{closure}(M))$.

In an applied safe lambda calculus, the Γ-variables with the lowest order must be all abstracted at once when applying the abstraction rule. Since the computation tree merges consecutive abstractions into a single node, any Γ-variable x occurring free in the subtree rooted at a λ-node $\lambda\overline{\xi} \notin IN^{\Sigma\vdash}$ different from the root must have order greater or equal to $\operatorname{ord} \lambda\overline{\xi}$. Conversely, if a lambda node $\lambda\overline{\xi}$ binds a variable node x then its order is $1 + \max_{z\in\overline{\xi}} \operatorname{ord} z > \operatorname{ord} x$.

Let x be a Γ-variable node in $\tau(\mathsf{closure}(M))$. Its enabler necessarily occurs in the path to the root, therefore, according to the previous observation, x must be bound by the first λ-node occurring in $[\circledast, x] \backslash IN^{\Sigma\vdash}$ with order strictly greater than $\operatorname{ord} x$. Hence τ is incrementally-bound.

(ii) We first show the result for closed terms. Let $\vdash M : T$ be a closed term such that $\tau(M)$ is incrementally-bound. We assume that M is already in η-long normal form. We prove by induction that M is safe. The base case $M \equiv \lambda\overline{\xi}.\alpha$ for some variable or constant α is trivial. *Step case:* $M \equiv \lambda\overline{\xi}.N_1 \ldots N_p$. Let $1 \leq i \leq p$. Each N_i can be written $\lambda\overline{\eta_i}.N'_i$ where N'_i is not an abstraction. By the induction hypothesis, $\lambda\overline{\xi}.N_i \equiv \lambda\overline{\xi}\overline{\eta_i}.N'_i$ is safe which means that the term N'_i is also safe: we have $\overline{\xi}, \overline{\eta_i} \vdash_{\mathsf{s}} N'_i : A_i$ for some type A_i. Let z be a variable occurring free in N'_i. Since M is closed, z is either bound by $\lambda\overline{\eta_1}$ or $\lambda\overline{\xi}$. In the latter case, since $\tau(M)$ is i.b. we have that $\operatorname{ord} z$ is smaller than $\operatorname{ord} \lambda\overline{\eta_1} = \operatorname{ord} N_i$, thus in both case we are allowed to abstract the variables $\overline{\eta_1}$ using the rule (abs), which shows that N_i is safe. Since all the N_is are safe and the term $N_1 \ldots N_p : o$ is of order 0, by the rule (app) we have that $N_1 \ldots N_p$ is safe: $\overline{\xi} \vdash_{\mathsf{s}} N_1 \ldots N_p : o$. The rule (abs) then gives us the sequent $\vdash_{\mathsf{s}} \lambda\overline{\xi}.N_1 \ldots N_p : T$.

Now if M is open, by the preceding case we have that $\mathsf{closure}(M)$ is safe. By "peeling-off" abstractions from a safe term we obtain an almost safe term, thus M is almost safe. $\square$

Note that the hypothesis that M is closed in (ii) is necessary. Take for instance the two terms $\lambda xy.x$ and $\lambda y.x$, where $x, y : o$. They have isomorphic incrementally-bound computation trees, but $\lambda xy.x$ is safe and $\lambda y.x$ is only almost safe.

For the second part of this proposition, a slightly stronger result holds if the term is β-normal and does not contain any interpreted constant:

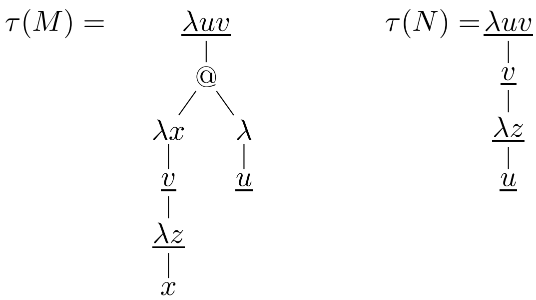
Corollary 5.12. *Let M be a β-normal term containing no interpreted constant. If all the* input variables *are incrementally-bound then the η-long normal form of M is almost safe, and safe if further M is closed.*

This is simply because in the computation tree of such terms all the variable nodes are *input*-variable nodes. This stronger result does not hold for terms containing redexes: for every unsafe closed term U, the term $(\lambda u.u)\ U$ is unsafe but the only input-variable is u and it is incrementally-bound. It does not hold either for terms with interpreted constants: for every closed unsafe term U of type `exp`, the PCF term `succ` U has no input variable but it is unsafe.

Corollary 5.13. *If $\tau(M)$ is incrementally-bound and $M \rightarrow_{\beta_s} N$ then $\tau(N)$ is incrementally-bound.*

Proof. Suppose that $\tau(M)$ is i.b. Then by Proposition 5.11(ii) the eta-long normal form of M is almost safe, therefore so is M by Lemma 3.41. But almost safety is preserved by β_s-reduction (Lemma 3.42) therefore N is almost safe, and by Proposition 5.11(i), $\tau(N)$ is incrementally-bound. $\square$

Note that this corollary cannot be generalized to A-incremental-binding for every set of nodes A. Take for instance the term $M \equiv \lambda u^o v^{((o,o),o)}.(\lambda x^o.v(\lambda z^o.x))u$ which beta-reduces to $N \equiv \lambda uv.v(\lambda z.u)$. The computation trees are:



If we take A to be the set of nodes that are hereditarily enabled by the root (underlined in the figure above) then $\tau(M)$ is A-incrementally-bound but $\tau(N)$ is not.

Incremental justification and incremental binding

Proposition 5.14 (Incremental-binding and P-incremental justification)**.** *Let $\Gamma \vdash M : T$ be a term-in-context of some applied typed lambda calculus.*

(i) Suppose M is β-normal. If all the reachable *input-variable nodes of the computation tree $\tau(\Gamma \vdash M : T)$ are incrementally bound then $[\![\Gamma \vdash M : T]\!]$ is P-incrementally justified.*

(ii) If $[\![\Gamma \vdash M : T]\!]$ is P-incrementally justified then all the reachable *input-variable nodes of the computation tree $\tau(\Gamma \vdash M : T)$ are $IN^{\circledast\vdash}$-incrementally bound.*

Proof. (i) Suppose M is a β-nf. W.l.o.g. we can assume that M is a closed term since the incremental-binding property is conserved when taking the closure of a term and since the denotation of the closure is isomorphic to the denotation of the term.

Suppose that all the reachable input-variable nodes of $\tau(M)$ are incrementally bound. We want to show that $[\![M]\!]$ is P-incrementally justified. Take a play $s \in [\![M]\!]$ ending with a question

P-move q. By the Correspondence Theorem 4.96, there is a traversal t of $\tau(M)$ starting with an occurrence r of the root $\circledast$ such that $\psi_M(t \upharpoonright r) = s$. We assume t to be the shortest such traversal, so that the last occurrence of t—name it n—is hereditarily justified by r, and is by definition an occurrence of a reachable node. Since ψ_M maps n to the P-question q, n is necessarily an occurrence of a variable node x. By Lemma 4.84 (iv), the P-views of s and $t \upharpoonright r$ are computed identically and have the same underlying sequence of justification pointers so in particular the node n and the move q both point to the same position in the justified sequence $\ulcorner t \upharpoonright r \urcorner$ and $\ulcorner s \urcorner$ respectively. Further by Lemma 4.84(iii), ψ_M maps nodes of a given order to moves of the same order. Hence showing that s is P-incrementally justified amounts to showing that n's justifier in t is the latest lambda node in $\ulcorner t \upharpoonright r \urcorner$ with order strictly greater than $\operatorname{ord} n$.

Let m denote n's justifier in t. The term M is closed therefore x is necessarily a bound variable and n is an occurrence of x's binder in $\tau(M)$. The traversal t is incrementally-bound by assumption and n belongs to $IN \setminus IN^{\Sigma\vdash} = IN^{\circledast\vdash}$ therefore by definition of incremental binding the occurrence m is the last λ-node in $[\circledast, n] \cap IN^{\circledast\vdash}$ with order strictly greater than $\operatorname{ord} n$. The Path–P-view correspondence (Prop. 4.29) gives $[\circledast, n] \cap IN^{\circledast\vdash} = \ulcorner t \urcorner \upharpoonright r$ which in turn equals $\ulcorner t \upharpoonright r \urcorner$ by Lemma 4.66 (It is applicable because M is a β-nf and we have assumed that the constant traversals are well-behaved).

(ii) Suppose $[\![M]\!]$ is P-incrementally justified. Let x be a reachable input-variable node of $\tau(M)$. There exists a traversal of the form $t \cdot x$ in $\mathcal{T}rav(M)$ such that x is hereditarily justified in t by the first occurrence r of $\tau(M)$'s root.

The correspondence theorem shows that $\varphi((t \cdot x) \upharpoonright r) = \varphi(t \upharpoonright r) \cdot \varphi(x)$ belongs to $[\![M]\!]$. Since $[\![M]\!]$ is P-incrementally justified, $\varphi(x)$ points to the last O-move in $\ulcorner \varphi(t \upharpoonright r) \urcorner$ with order strictly greater than $\operatorname{ord} \varphi(x)$. Consequently x points to the last λ-node in $\ulcorner t \upharpoonright r \urcorner$ with order strictly greater than $\operatorname{ord} x$.

But by Lemma 4.65, $\ulcorner t \upharpoonright r \urcorner$ contains $\ulcorner t \urcorner \upharpoonright r$ as a subsequence, and by P-visibility m occurs in this subsequence, thus m is also the last λ-node in $\ulcorner t \urcorner \upharpoonright r$ with order strictly greater than $\operatorname{ord} x$. By the Path-P-view correspondence (Prop. 4.29) this means that m is the last λ-node in $[\circledast, x[\cap IN^{\circledast\vdash}$ with order strictly greater than $\operatorname{ord} x$. Hence $\tau(M)$ is $IN^{\circledast\vdash}$-incrementally-bound. ◻

Corollary 5.15. *Let $\Gamma \vdash M : A$ be a term-in-context of some applied typed lambda calculus.*

(i) If $\tau(\Gamma \vdash M : A)$ is incrementally-bound then $[\![\Gamma \vdash M : A]\!]$ is P-incrementally justified;

(ii) if M is β-normal and $[\![\Gamma \vdash M : A]\!]$ is P-incrementally justified then $\tau(\Gamma \vdash M : A)^*$ is incrementally-bound.

Proof. (i) Let M' denote the beta-normal form of M. If $\tau(M)$ is incrementally bound then by Corollary 5.13 so is $\tau(M')$. So in particular all the *reachable* input-variable node of $\tau(M')$ are incrementally bound. Thus by Proposition 5.14(i), $[\![M]\!] = [\![M']\!]$ is P-incrementally justified.

(ii) Suppose that $[\![M]\!]$ is P-incrementally justified. Consider $\tau(M)^*$. By definition, a tree is incrementally bound just if it is $N \setminus IN^{\Sigma\vdash}$-incrementally bound. Since M is β-normal, variable nodes cannot be hereditarily enabled by an @-node thus $IN^{\vdash\circledast} = N \setminus IN^{\Sigma\vdash}$. Thus to show that $\tau(M)^*$ is incrementally-bound we just need to show that its variables are $IN^{\vdash\circledast}$-incrementally bound. But by definition its variable nodes are precisely those of $\tau(M)$ that are reachable. Hence we just need to show that the reachable input variables of $\tau(M)$ are $IN^{\vdash\circledast}$-incrementally bound. This is precisely what Proposition 5.14(ii) says. ◻

5.4 Safe lambda calculus

We now consider the special case of the pure (*i.e.*, without interpreted constants) safe lambda calculus. For every simply-typed term $\Gamma \vdash_{\mathsf{st}} M : T$ we write $[\![\Gamma \vdash_{\mathsf{st}} M : T]\!]$ to refer to the innocent game denotation of $\Gamma \vdash_{\mathsf{st}} M : T$.

Lemma 5.16. *Let M be a simply-typed lambda-term in β-normal form. All the nodes of the computation tree of M are* reachable *by some traversal obtained using the rules of Table 4.3.*

Proof. Since M is in β-normal form, its computation tree has no application node and therefore all the variable nodes are hereditarily justified by the root. Hence each variable node can be reached by the traversal consisting of the path from the root to that node (The rule (Lam) and $(\mathsf{InputVar})$ permit us to visit the variable nodes and lambda nodes respectively). ☐

Proposition 5.17. *Let $\Gamma \vdash_{\mathsf{st}} M : T$ be a pure (i.e., with no interpreted constants) simply-typed term in β-normal form. Then $[\![\Gamma \vdash_{\mathsf{st}} M : T]\!]$ is P-incrementally justified if and only if the computation tree $\tau(M)$ is incrementally-bound.*

Proof. By Lemma 5.16, all the variable nodes are reachable in a β-normal term thus $\tau(M) = \tau(M)^*$ and the result follows from Corollary 5.15. ☐

Example 5.18.

(i) For every higher-order variable $x : A$ the computation tree $\tau(x)$ is incrementally-bound. Consequently the projection strategies are all P-incrementally justified.

(ii) Consider the β-normal term $\Gamma \vdash_{\mathsf{st}} f(\lambda y^o.x) : o$ where $\Gamma = f : 2,\ x : o$. The figure on the right represents its computation tree with the node orders given as superscripts. The node x is not incrementally-bound because the node x of order 0 is not bound by the first-order node λy. Therefore $\tau(f(\lambda y^o.x))$ is not incrementally-bound and by Proposition 5.17, $[\![\Gamma \vdash_{\mathsf{st}} f(\lambda y^o.x) : o]\!]$ is not P-incrementally justified. Similarly we can check that $[\![\lambda y^o.x]\!]$ is P-i.j. while $[\![f(\lambda y^o.x)]\!]$ is not.

$$\begin{array}{c} \lambda^3 \\ | \\ f^2 \\ | \\ \lambda y^1 \\ | \\ x^0 \end{array}$$

(iii) By the previous examples we have that $[\![\Gamma \vdash_{\mathsf{st}} f : 2]\!]$ and $[\![\Gamma \vdash_{\mathsf{st}} \lambda y^o.x : 1]\!]$ are both P-i.j. whereas $[\![\Gamma \vdash_{\mathsf{st}} f(\lambda y^o.x) : o]\!]$ is not. Hence application does not preserve P-incremental justification. This suggests that P-incremental justification is not a compositional property. In Chapter 6 we will identify a sufficient condition enabling compositionality of P-incrementally justified strategies.

Putting Proposition 5.17 and Proposition 5.11 together gives us a game-semantic characterization of safety. This result was first presented in TLCA2007, [BO07, Theorem 3(ii)]:

Theorem 5.19 (Characterization Theorem for the safe lambda calculus)**.** *Let $\Gamma \vdash_{\mathsf{st}} M : A$ be a pure simply-typed term (with no interpreted constants).*

(i) If M is almost safe *(and in particular if it is safe) then $[\![\Gamma \vdash_{\mathsf{st}} M : A]\!]$ is P-incrementally justified.*

(ii) If $[\![\Gamma \vdash_{\mathsf{st}} M : A]\!]$ is P-incrementally justified then the beta-normal form of M is almost safe, and safe if further M is closed.

Proof. (i) Since M is almost safe, by Proposition 5.11(i), its computation tree is incrementally-bound. Hence by Corollary 5.15(i) its denotation is incrementally justified.

(ii) Since a term has the same denotation as its beta-normal form we can assume that M is beta-normal. By Proposition 5.17 its computation tree is incrementally bound, and by Proposition 5.11(ii), the eta-long normal form of M is safe if it is a closed term and almost safe otherwise. The same holds for M itself since both safety and unsafety are preserved by eta-long normal expansion (Lemma 3.41 and 3.37). ☐

In particular, a term has a P-incrementally justified denotation if and only its beta-normal form is almost safe.

Remark 5.20

(i) Observe that the use of the Correspondence Theorem makes the proof of the above theorem almost trivial: just by making some observations about the computation trees of safe terms, we are able to deduce properties in the denotational game model. We do not claim here that it is the unique way to prove the result, however any proof would require at some point to make a connection between the binding information found in the syntax of the term, and the justification pointers of game semantics. In our argument, this connection is provided by the concrete presentation of game semantics from the previous chapter.

(ii) In game semantics, the Opponent's strategy is dictated by the denotation of a term—the context—representing the environment so that if the language considered is a pure functional language such as PCF then the Opponent necessarily plays innocently. In the intensional game denotation, however, all possible O-moves are accounted for at every position, including those moves that would break "O-innocence". In the extensional denotation, non O-innocent plays do not have any effect since the test strategy from the intrinsic preorder ranges over *P-innocent* strategies only.

The second part of the previous theorem crucially relies on the presence of those non O-innocent plays: It is true that an unsafe beta normal term is denoted by a non P-i.j. strategy, but the failure to satisfy P-incremental justification may only be due to some play that does not affect the extensional denotation of the term. For instance the beta-normal term $\lambda\varphi^{((o,o),o,o)}\, y^o.\ \varphi(\lambda x^o.x)(\varphi(\lambda x^o.\underline{y})\, y)$ is clearly unsafe and, as is implied by (ii), its denotation in the intensional game model is not P-i.j. since for instance the last node in the traversal $t = \lambda\varphi y \cdot \varphi^1 \cdot \lambda \cdot \varphi^2 \cdot \lambda x \cdot y$ is not incrementally justified. But the traversal t corresponds to a play that does not respect O-innocence: we have $\llcorner t_{\leqslant \varphi^1} \lrcorner = \llcorner t_{\leqslant \varphi^2} \lrcorner$ but the nodes φ^1 and φ^2 are followed by two different nodes.

Putting Theorem 5.19(i) and Lemma 5.2 together gives:

Proposition 5.21 (P's pointers are superfluous for safe terms)**.** *In the game semantics of safe lambda-terms, pointers emanating from P-moves are unnecessary: they are uniquely recoverable from the underlying sequences of moves and from O-moves' pointers.*

Example 5.22. If justification pointers are omitted then the denotations of the two Kierstead terms $M_1 \equiv \lambda f^2.f(\lambda x^o.f(\lambda y^o.y))$ and $M_2 \equiv \lambda f^2.f(\lambda x^o.f(\lambda y^o.x))$ from Example 3.5 are not distinguishable. In the safe lambda calculus this ambiguity disappears since M_1 is safe whereas M_2 is not (The free variable x in the subterm $f(\lambda y^o.x)$ has the same order as y but it is not abstracted together with y).

In fact, as the last example highlights, pointers are superfluous at order 3 for safe terms whether from P-moves or O-moves. This is because for question moves in the first two levels of an arena (initial moves being at level 0), the associated pointers are uniquely recoverable thanks to the visibility condition. At the third level, the question moves are all P-moves therefore their associated pointers are uniquely recoverable by P-incremental justification. This is not true anymore at order 4: Take the safe term-in-context $\psi : (((o^4, o^3), o^2), o^1) \vdash_{\mathsf{s}} \psi(\lambda\varphi^{(o,o)}.\varphi\, a) : o^0$ for some constant $a : o$. Its strategy denotation contains plays whose underlying sequence of moves is $q_0\, q_1\, q_2\, q_3\, q_2\, q_3\, q_4$. Since q_4 is an O-move, it is not constrained by P-incremental justification and thus it can point to any of the two occurrences of q_3.[2]

[2]More generally, a P-incrementally justified strategy can contain plays that are not "O-incrementally justified" since it must take into account any possible strategy incarnating its context, including those that are not P-incrementally justified. For instance in the given example, there is one version of the play that is not O-incrementally justified (the one where q_4 points to the first occurrence of q_3). This play is involved in the strategy composition $[\![\vdash_{\mathsf{st}} M_2 : (((o,o),o),o)]\!]; [\![\psi : (((o,o),o),o) \vdash_{\mathsf{st}} \psi(\lambda\varphi^{(o,o)}.\varphi a) : o]\!]$ where M_2 denotes the unsafe Kierstead term.

5.5 Safe PCF

We now extend the game-semantic characterization to safe PCF. We have already established the correspondence between almost safety and incremental binding in the general setting of an applied simply-typed lambda calculus without recursion (Proposition 5.11). PCF_1 can be cast into this setting by considering the special constant node $\perp$, that represents the non-terminating term Ω_o in the computation tree, as an ordinary constant. In full PCF, however, a difficulty arises as computation trees are potentially infinite due to the presence of the Y combinator. Nevertheless the result still holds:

Proposition 5.23 (Almost safety and incrementally-binding). *Let $\Gamma \vdash M : A$ be a PCF term.*

(i) If $\Gamma \vdash M : A$ is almost safe then $\tau(\Gamma \vdash M : A)$ is incrementally-bound ;

(ii) conversely, if $\tau(\Gamma \vdash M : A)$ is incrementally-bound then the η-long normal form of $\Gamma \vdash M : A$ is almost safe if M is open and safe if M is closed.

Proof. (i) Let M be an almost safe PCF term and i denote the number of occurrences of the Y combinator in M. We first prove by induction on i that for every $k \in \omega$, the k^{th} approximants to M, denoted M_k, is almost safe. The base case $i = 0$ is trivial: $M_k = M$. Step case: $i > 0$. Let $\mathsf{Y}_A N$ be a subterm of M. Since M is almost safe, N is also safe. The number of occurrences of the Y combinator in N is smaller than i therefore by the induction hypothesis N_k is safe. Consequently the term $\mathsf{Y}^k_A N_k = \underbrace{N_k(\ldots(N_k}_{k \text{ times}}\Omega)\ldots)$ is also safe and by compositionality so is M_k.

The result holds for PCF_1 terms, thus since M_k is a safe PCF_1 term, $\tau(M_k)$ is incrementally-bound. Now let z be a variable node in $\tau(M) = \bigcup_{k\in\omega} \tau(M_k)$. There exists $k \in \omega$ such that z belongs to $\tau(M_k) \sqsubseteq \tau(M)$. If we write r_k to denote the root of the tree $\tau(M_k)$ then the path $[r_k, z]$ in $\tau(M_k)$ is equal to the path $[r, z]$ in $\tau(M)$. Hence, since z is incrementally-bound in $\tau(M_k)$, it is also incrementally-bound in $\tau(M)$.

(ii) Suppose that the term is not almost safe then necessarily one of its approximant is not almost safe either. Since the result holds for every PCF_1 term, the computation tree of the approximant is not incrementally-bound. But the computation tree of M contains the computation tree of its approximant, therefore it is not incrementally-bound. ☐

We thus obtain the following characterization of almost safety by P-incrementally justified strategies:

Theorem 5.24 (Characterization Theorem for safe PCF). *Let $\Gamma \vdash M : A$ be a PCF term. Then:*

(i) If M is almost safe then $[\![\Gamma \vdash M : A]\!]$ is P-incrementally justified.

(ii) If $[\![\Gamma \vdash M : A]\!]$ is P-incrementally justified then $\eta_{\mathsf{lnf}}(\beta_{\mathsf{nf}}(M))^*$ is almost safe if M is open, and safe if M is closed.

Proof. (i) Let M be an almost safe term and M^∞ be the β-normal form of M. Since almost-safety is preserved by the small-step reduction of PCF, M^∞ is also almost-safe and by Proposition 5.23, $\tau(M^\infty)$ is incrementally-bound. By Corollary 5.15(i), $[\![M^\infty]\!]$ is P-incrementally justified and by soundness of the game denotation, $[\![M^\infty]\!] = [\![M]\!]$, thus $[\![M]\!]$ is P-incrementally justified.
(ii) Let M be PCF term with a P-incrementally justified denotation. By Corollary 5.15(ii), $\tau(\beta_{\mathsf{nf}}(M))^* = \tau(\eta_{\mathsf{lnf}}(\beta_{\mathsf{nf}}(M))^*)$ is incrementally-bound. Hence by Proposition 5.23(ii), if M is closed then $\eta_{\mathsf{lnf}}(\beta_{\mathsf{nf}}(M))^*$ is safe and almost safe otherwise. ☐

Consequently, P-pointers are superfluous (*i.e.*, uniquely recoverable) in the game denotation of safe PCF terms.

Example 5.25 (Counter-example)**.** The use of dead-code elimination in the second part of the theorem is crucial. Take for instance the unsafe closed PCF term:

$$M \equiv \lambda f^{((\mathtt{exp},\mathtt{exp}),\mathtt{exp})}\, x^{\mathtt{exp}}\, y^{\mathtt{exp}}.f(\lambda z^{\mathtt{exp}}.\mathtt{cond}(\mathtt{succ}\,\underline{x})\,\underline{y}\,z)\ .$$

This term is in β-normal form (the conditional operator cannot be reduced since the value of x is undetermined). The η-long β-normal form of M is therefore M itself which is unsafe. But since $\mathtt{succ}\,x$ always evaluate to a positive integer, the first branch of the conditional operator will never be evaluated. Hence M is observationally equivalent to the safe term $N \equiv \lambda fxy.f(\lambda z.z)$ which by soundness of the game model implies that they have the same denotation. Therefore since N is safe, by the first part of the theorem we have that $[\![M]\!]$ is P-incrementally justified.

Such counter-example arises because the conditional operator of PCF permits us to construct beta-normal terms containing "dead code" (*i.e.*, some subterm that will never be evaluated for every value of M's parameters). In the example above, the dead code consists of the subterm y. In general, if the dead code part of the computation tree contains a variable that is not incrementally bound then the resulting term will be unsafe even if the rest of the tree is incrementally bound. Here it is possible to turn M into the equivalent safe term N by eliminating the dead code from M.

5.6 Safe Idealized Algol

The argument used in the previous section for safe PCF can be reused identically for safe IA (as defined in Sec. 3.5.2.2). Hence we have:

Theorem 5.26 (Characterization Theorem for Safe IA)**.** *Let $\Gamma \vdash M : A$ be an IA term. Then:*

(i) If M is almost safe then $[\![\Gamma \vdash M : A]\!]$ is P-incrementally justified.

(ii) If $[\![\Gamma \vdash M : A]\!]$ is P-incrementally justified then $\eta_{\mathsf{lnf}}(\beta_{\mathsf{nf}}(M))^*$ is almost safe if M is open, and safe if M is closed.

This shows that P-pointers are superfluous for safe IA terms. Since unsafety only appears at order 3, this theorem implies the well-known result that pointers are uniquely recoverable for IA_2 terms. This suggests potential applications in algorithmic game semantics: Ghica and McCusker were able to show that the game denotation of IA_2 terms can be characterized by (extended) regular expressions, thus giving a decision procedure for observational equivalence in this fragment [GM00]. Can we obtain a similar result for higher-order fragment of safe IA? We will investigate this question in the next chapter.

5.7 Towards a game model of safe PCF

5.7.1 Definability

Recall (Sec. 2.3.4.6) that PCF_c denotes the language obtained by extending PCF with the $\mathtt{case}_k$ construct—the obvious generalization of the conditional operator $\mathtt{cond}$ to $k > 2$ branches instead of 2. We call safe PCF_c the corresponding extension of safe PCF. Clearly, all the results obtained so far concerning safe PCF also hold in safe PCF_c.

The characterization theorem allows us to show the following definability result for safe PCF_c:

Proposition 5.27 (Definability for safe PCF_c terms)**.** *Let $\overline{A} = (A_1,\ldots,A_i)$ and B be two PCF types for $i \geq 0$ and σ be a well-bracketed innocent P-i.j. strategy with finite view function defined on the game $A_1 \times \ldots \times A_i \rightarrow B$. There exists an* almost safe *PCF_c term $\overline{x} : \overline{A} \Vdash_\sigma M : B$ in η-long normal form such that:*

$$[\![\overline{x} : \overline{A} \Vdash M_\sigma : B]\!] = \sigma$$

and a safe closed PCF_c term $\vdash_{\mathsf{s}} M'_\sigma : (\overline{A}, B)$ *in η-long normal form such that:*

$$[\![\vdash_{\mathsf{s}} M'_\sigma : (\overline{A}, B)]\!] \cong \sigma \ .$$

Proof. By the standard definability result for PCF_c, there is a *finite* term-in-context $\overline{x} : \overline{A} \vdash M_\sigma : B$ such that $[\![\overline{x} : \overline{A} \vdash M_\sigma : B]\!] = \sigma$. An inspection of the proof [AMJ94, HO00] shows that M_σ is by construction in beta-eta normal form and further it contains no dead-code. Hence by Theorem 5.24(ii), M_σ is almost safe. For the second part take $M'_\sigma \equiv \lambda \overline{x}^{\overline{A}}.M_\sigma$. ☐

This result (that every *compact* strategy with finite view function is definable) is often summarized by saying that the game model is ***intensionally fully-abstract*** [AMJ94].

5.7.2 Compositionality

In the next chapter we will give an in depth account of P-i.j. strategies. In particular we will give a semantic argument showing that when suitably restricted, P-i.j. strategies compose. We show here essentially the same result using a syntactic argument that relies on the definability result from the previous paragraph. The advantage is that the proof is much simpler than the one given in the next chapter. The disadvantage is that it is slightly less general as it only works for strategies that are denotations of compact PCF terms (*i.e.*, the compact innocent ones) whereas the semantic proof from the next chapter works in the general case.

Problem: Let $\overline{A} = (A_1, \ldots, A_i)$, $B = (B_1, \ldots, B_l, o)$ and $C = (C_1, \ldots, C_k, o)$ be three PCF types for some $i \geq 1, l, k \geq 0$. Given two compact (with finite view function) innocent well-bracketed and P-incrementally justified strategies $f : \ A_1 \times \ldots \times A_i \rightarrow B$ and $g : \ B \rightarrow C$. Which condition shall we impose for the composite $f;g$ to be P-incrementally justified?

A sufficient and necessary condition We tackle the problem syntactically by appealing to the definability result: Since f and g are compact innocent, there are two closed safe terms $M_f : (\overline{A}, B)$ and $M_g : B \rightarrow C$ in η-long nf denoted by f and g respectively. Composition is syntactically formulated by the term

$$M_{f;g} \equiv \lambda \overline{x}^{\overline{A}}.M_g(M_f \overline{x})$$

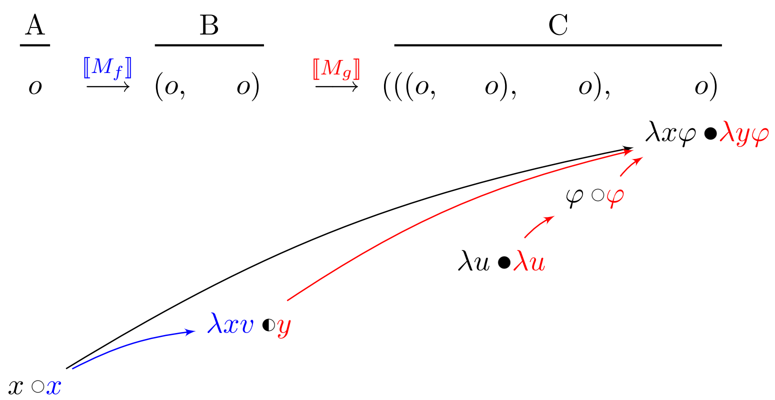
for some fresh variables $\overline{x} : \overline{A}$. Indeed its denotation is:

$$[\![M_{f;g}]\!] = [\![M_f]\!];[\![M_g]\!] = f;g \ . \tag{5.2}$$

Observe that the safety of M_f and M_g does not imply that of $M_{f;g}$ as the following examples illustrate:

Example 5.28. (i) Take the types $A = o$, $B = 1$, $C = 3$ and closed safe terms $M_f \equiv \lambda x^o v^o.x : A \rightarrow B$ and $M_g \equiv \lambda y^B \varphi^2.\varphi(\lambda u^o.ya) : B \rightarrow C$ for some Σ-constant $a : o$. . The eta-long beta-nf of $M_{f;g}$ is $\lambda x^o \varphi^2.\varphi(\underline{\lambda u^o.x})$ which is unsafe because of the underlined term.

Consequently by Theorem 5.19(ii), the strategy $[\![M_{f;g}]\!] = [\![M_f]\!];[\![M_g]\!]$ is not P-i.j. This shows that P-i.j. strategies do not generally compose. The following diagram represents a play that is not P-i.j.:



(ii) A counter-example with $\operatorname{ord} B = \operatorname{ord} C$: Take the types $A = o$, $B = C = 3$ and closed safe terms $M_f \equiv \lambda x^A v^2.x$ and $M_g \equiv \lambda y^B \varphi^2.\varphi(\lambda u^o.y(\lambda g^1.a))$ for some Σ-constant $a : o$. The $\eta\beta$-nf of $M_{f;g}$ is $\lambda x^A \varphi^2.\varphi(\underline{\lambda u^o.x})$ which is unsafe because of the underlined term, so $f;g$ is not P-i.j.

Since M_f and M_g are in η-nf, they can be written:

$$\begin{aligned} \vdash_{\mathsf{s}} M_f &\equiv \lambda x_1^{A_1} \ldots x_i^{A_i}\ \varphi_1^{B_1} \ldots \varphi_l^{B_l}.N_f : A \rightarrow B \\ \vdash_{\mathsf{s}} M_g &\equiv \lambda y^{(B_1,\ldots,B_l,o)}\ \phi_1^{C_1} \ldots \phi_k^{C_k}.N_g : B \rightarrow C \end{aligned}$$

for some safe ground-type terms N_f and N_g in η-nf. Substituting these two terms in $M_{f;g}$ in (5.2) gives:

$$\begin{aligned} f;g &= [\![\lambda \overline{x}^{\overline{A}}.(\lambda \phi_1^{C_1} \ldots \phi_k^{C_k}.N_g)[(M_f \overline{x})/y]]\!] \\ &= [\![\lambda \overline{x}^{\overline{A}} \phi_1^{C_1} \ldots \phi_k^{C_k}.N_g[(M_f \overline{x})/y]]\!] \qquad \text{(the } x_j\text{'s and } \phi_j\text{'s can be chosen to be disjoint).} \end{aligned} \tag{5.3}$$

Thus by Theorem 5.24, $f;g$ is P-incrementally justified just when $\eta_{\mathsf{lnf}}(\beta_{\mathsf{nf}}(N_g[(M_f\overline{x})/y]))^*$ is safe.

Lemma 5.29. *Let $\Gamma, y : B \vdash_{\mathsf{s}} M : A$ be a safe term-in-context in η-nf and $\Gamma \vdash R : B$ be an almost safe application. Let IN denote the set of inner-nodes of the computation tree of M and $\circledast$ be the root. Then:*

$$\Gamma \vdash_{\mathsf{s}} M[R/y] : A \iff \forall x \in FV(R).\forall y \in IN_{\mathsf{fv}}.\forall m \in IN_{\lambda} \cap]\circledast, y] : \operatorname{ord} m \leq \operatorname{ord} x \ .$$

Proof. The only unsafety that can occur when substituting the almost safe term R for y in M is when some free variable in R is not incrementally bound in $\tau(M)$. The right-hand side of the equivalence expresses just this. ☐

Applying this lemma with $R \equiv M_f \overline{x}$ and $M \equiv M_g$ gives us a necessary and sufficient condition for $M_g[(M_f\overline{x})/y]$ to be safe, and hence for $f;g$ to be P-i.j (provided that the term does not contain dead code). The fact that this condition is expressed on both M_g and M_f at the same time rather than independently is unsatisfactory because it does not give rise to a categorical notion of compositionality (Two morphisms should be composable as soon as the domain of one matches with the codomain of the other).

A sufficient condition In order for our P-i.j. strategies to compose in the categorical sense we need to add a further restriction: we consider strategies on games of the form $A_1 \times \cdots \times A_i \rightarrow B$ where each A_i denotes a PCF type and such that $\operatorname{ord} A_i \geq \operatorname{ord} B$ for $1 \leq i \leq n$. Strategies satisfying this condition are called *closed P-incrementally justified strategies*, they will be studied in depth in Sec. 6.2.4.

Lemma 5.30. *If* $\mathrm{ord}\, A_i \geq \mathrm{ord}\, B$ *for all* $1 \leq i \leq n$ *then* $f; g$ *is P-incrementally justified.*

Proof. For all $1 \leq i \leq n$ we have $\mathrm{ord}\, x_i = \mathrm{ord}\, A_i \geq \mathrm{ord}\, B = \mathrm{ord}\,(M_f \overline{x})$ thus we can use the application rule of the safe lambda calculus to form the safe term $\overline{x} : \overline{A} \vdash_{\mathsf{s}} M_f \overline{x} : B$. The substitution lemma shows that $M_g[(M_f \overline{x})/y]$ is safe which by (5.3) implies that $f; g$ is P-i.j. $\square$

Remark 5.31

1. The above condition is not necessary: Take $A = o$, $B = (o, o)$, $C = (o, o)$ and consider the two safe terms $M_f \equiv \lambda x^A u^o.u$ and $M_g \equiv \lambda y^B.y\, a$ for some constant $a : o$. Then we have $M_{f;g} =_\beta \lambda x^A.a$ which is safe hence $f; g$ is P-i.j. although $\mathrm{ord}\, A < \mathrm{ord}\, B$.

2. In general type homogeneity is not preserved after composition. For instance the types $o \to (o \to o)$ and $(o \to o) \to ((o \to o) \to o)$ are homogeneous but $o \to ((o \to o) \to o)$ is not. Incidentally, the condition of Lemma 5.30 turns out to be a sufficient condition for type-homogeneity to compose: if $A \to B$ and $B \to C$ are homogeneous simple types and $\mathrm{ord}\, A \geq \mathrm{ord}\, B$ then $A \to C$ is homogeneous.

5.7.3 Full abstraction

The definability result that we have shown for safe PCF is a first step towards full-abstraction. In Chapter 2 we have presented the full abstraction result for PCF [AMJ94, HO00, Nic94]: two PCF terms are observationally equivalent if and only they have the same denotation in the game model. Since safe PCF is a fragment of PCF this statement also holds for safe PCF terms: Two safe PCF terms are observationally equivalent *with respect to PCF contexts* (not necessarily safe) if and only if they have the same game denotation. A natural question is whether there exists a fully abstract model with *respect to safe contexts* only. Since safe PCF terms are denoted by P-incrementally justified strategies, it is reasonable to think that O-moves also need to be constrained by a symmetrical notion of "O-incremental justification" corresponding to the requirement that contexts are safe. We will show how this can be done in the next chapter.

Chapter 6

Models of Safe Applied Lambda Calculi

This chapter aims to formally define the notion of *model* of the safe lambda calculus and its various extensions. We present a categorical interpretation of the safe lambda calculus in the same vein as the characterization of the simply-typed lambda calculus by Cartesian Closed Categories. We then provide such a model by means of game semantics and show that it is fully-abstract when observational equivalence is defined with respect to safe contexts. We conclude the chapter by examining the model from an algorithmic game-semantic point of view: we consider the problem of observational equivalence for finitary fragments of safe IA and show that up to order 3, the complexity of deciding observational equivalence is essentially the same as for unrestricted IA terms. We then give a version of the complete-play Characterization Theorem for safe terms: we show that two safe terms are observationally equivalent if and only if the sets of complete O-incrementally justified plays of the denotations are equal. This result leads us to conjecture that observational equivalence is decidable for safe IA up to order 4.

6.1 Categorical model

It is well-known [Lam86] that cartesian closed categories (categories with a terminal object, finite products and exponentials), CCCs for short, capture the notion of model of typed lambda calculi: Every CCC is a model of the simply-typed lambda calculus, and conversely, every typed lambda calculus generates a CCC. *What is the categorical interpretation of the safe lambda calculus?* This section introduces *incremental closed categories* and shows that they capture models of safe lambda calculi.

6.1.1 Safe lambda calculus with product

The safe lambda calculus defined in Chapter 3 does not have products. It is easy to add them to the language. The type grammar is given by:

$$T ::= \ B \mid T \rightarrow T \mid T \times T$$

for some set B of base types. The ***order*** of a type is defined by induction as follows:

- $\operatorname{ord}(B) = 0$ for every base type B,
- $\operatorname{ord}(A \times B) = \max(\operatorname{ord} A, \operatorname{ord} B)$,
- $\operatorname{ord}(A \rightarrow B) = \max(1 + \operatorname{ord} A, \operatorname{ord} B)$.

The typing system of the safe lambda calculus is then extended with three rules corresponding to pairing, first projection and second projection (respectively $(\times)$, (π_1) and (π_2) in Table 6.1). This suffices to add product constructs to the safe lambda calculus but there is now a little problem. Consider the following terms-in-context:

$$x : (o \rightarrow o) \times o \vdash_{\mathsf{st}} \lambda z^o . (\pi_2 x) \equiv M_1 : (o \rightarrow o)$$

$$x_1 : (o \to o), x_2 : o \vdash_{\mathsf{s}} \lambda z^o . \underline{x_2} \equiv M_2 : (o \to o) \ .$$

In any model of the lambda calculus, these two terms-in-context have isomorphic denotations, but M_1 is safe whereas M_2 is unsafe. Indeed, the side-condition of the abstraction rule only requires that the *variables* in the context have order greater than the order of the term, therefore M_2 is unsafe because it contains the free occurrence x_2. In M_1, however, x_1 and x_2 are combined into a single variable, this has the effect of increasing the order of the variable and therefore the side-condition holds.

In the categorical model of the simply-typed lambda calculus, a term-in-context $\Gamma \vdash M : T$ is modeled by a morphism $[\![\Gamma]\!] \to [\![T]\!]$ where the context Γ is identified with the product of the types of the variables in the context: if the context variables are $X_1, \cdots, X_n$ then Γ is identified with the type $X_1 \times \cdots \times X_n$. Thus the contexts $x_1 : A, x_2 : B$ and $x : A \times B$ will be denoted by two isomorphic objects in the category. Because variables in the context can be "combined", there is no way to tell—just by looking at the *type* Γ—which subtypes corresponds to which variable. Consequently the basic property of the safe lambda calculus—that all the variables in the context have order greater than the order of the term—cannot be expressed in the standard categorical model. For this reason we modify slightly the side-condition of the abstraction and application rules to enforce a property stronger than the usual basic property of the safe lambda calculus: instead of requiring that all variables in the context have order greater than the order of the term, we require that the order of *any prime sub-type of any variable* in the context has order greater than that of the term, where the set of ***prime sub-types*** of a type A, written $Pr(A)$, is given by:

$$\begin{aligned} Pr(B) &= \{B\} && \text{if } B \text{ is a base type,} \\ Pr(A \to B) &= \{A \to B\} \\ Pr(A \times B) &= Pr(A) \cup Pr(B) \ . \end{aligned}$$

We then define the relation $\geq$ on types as follows:

$$A \geq B \quad \stackrel{\text{def}}{=} \quad \forall A' \in Pr(A).\, \mathrm{ord}\, A' \geq \mathrm{ord}\, B \ .$$

Thus for every context Γ and type B we have:

$$\Gamma \geq B \iff \forall x : A \ \in \Gamma . \forall A' \in Pr(A).\, \mathrm{ord}\, A' \geq \mathrm{ord}\, B \ .$$

We now replace the side-condition in the abstraction and application rules by "$\Gamma \geq B$" where B denotes the type of the term being formed and Γ its context.

Definition 6.1. The ***safe lambda calculus with product***, or safe $\Lambda^{\times}_{\to}$ for short, over a typed-alphabet Ξ of constants is given by induction over the rules of Table 6.1. The differences with the rules of the safe lambda calculus without product are framed.

Example 6.2. The terms M_1 and M_2 given above are both unsafe.

It is easy to see that the basic property of the safe lambda calculus still holds—the free variables of a term have order greater than the order of the term itself—and therefore all the basic results showed in Chapter 3 also hold (No-variable-capture lemma, safety is preserved by safe β reduction, ...).

We call ***typed calculus*** any applied simply-typed lambda calculus with product with a stock of constants and function symbols together with an operational semantics for function symbols given by means of a set of reduction rules. We define the ***safe fragment*** of a typed calculus as the system obtained by replacing the abstraction and application rules by the rules (app), $(\mathsf{app_{as}})$, (abs) and (δ) from Table 6.1. A language that is the safe fragment of some typed lambda calculus is called a ***safe typed calculus***.

The ***long safe fragment*** of a type-calculus is the subclass of the safe fragment consisting of terms-in-context that are typable without using the rule $(\mathsf{app_{as}})$. (See Def. 3.31.)

$$(\mathsf{var})\ \frac{}{x : A \vdash_{\mathsf{s}} x : A} \qquad (\mathsf{const})\ \frac{}{\vdash_{\mathsf{s}} f : A}\ f \in \Xi \qquad (\mathsf{wk})\ \frac{\Gamma \vdash_{\mathsf{s}} s : A}{\Delta \vdash_{\mathsf{s}} s : A}\ \Gamma \subset \Delta \qquad (\delta)\ \frac{\Gamma \vdash_{\mathsf{s}} M : A}{\Gamma \Vdash_{\mathsf{app}} M : A}$$

$$\boxed{(\times)\ \frac{\Gamma \vdash_{\mathsf{s}} s : A \qquad \Gamma \vdash_{\mathsf{s}} t : B}{\Gamma \vdash_{\mathsf{s}} \langle s, t\rangle : A \times B} \qquad (\pi_1)\ \frac{\Gamma \vdash_{\mathsf{s}} s : A \times B}{\Gamma \vdash_{\mathsf{s}} \pi_1 s : A} \qquad (\pi_2)\ \frac{\Gamma \vdash_{\mathsf{s}} s : A \times B}{\Gamma \vdash_{\mathsf{s}} \pi_2 s : B}}$$

$$(\mathsf{app}_{\mathsf{as}})\ \frac{\Gamma \vdash_{\mathsf{s}} s : (A_1, \ldots, A_n, B) \quad \Gamma \vdash_{\mathsf{s}} t_1 : A_1 \quad \ldots \quad \Gamma \vdash_{\mathsf{s}} t_n : A_n}{\Gamma \Vdash_{\mathsf{app}} s\ t_1 \ldots t_n : B}$$

$$(\mathsf{app})\ \frac{\Gamma \vdash_{\mathsf{s}} s : (A_1, \ldots, A_n, B) \quad \Gamma \vdash_{\mathsf{s}} t_1 : A_1 \quad \ldots \quad \Gamma \vdash_{\mathsf{s}} t_n : A_n}{\Gamma \vdash_{\mathsf{s}} s\ t_1 \ldots t_n : B}\ \boxed{\Gamma \geq B}$$

$$(\mathsf{abs})\ \frac{\Gamma, x_1 : A_1, \ldots, x_n : A_n \Vdash_{\mathsf{app}} s : B}{\Gamma \vdash_{\mathsf{s}} \lambda x_1 \ldots x_n.s : (A_1, \ldots, A_n, B)}\ \boxed{\Gamma \geq (A_1, \ldots, A_n, B)}$$

Table 6.1: The safe lambda calculus with product (safe $\Lambda^{\times}_{\to}$).

Remark 6.3 (Alternative definition) Our definition of the safe lambda calculus with product conveys the syntactic notion of safety appropriately but there is still a mismatch between syntax and semantics: there exist pairs of terms, one safe and the other unsafe, that are denoted by the same (up to isomorphism) morphism in the categorical model of the simply-typed lambda calculus. For instance the two simply-typed terms-in-context:

$$x : (o \to o) \times o \vdash_{\mathsf{st}} \lambda z^o.\, (\pi_1 x) \equiv N_1 : (o \to (o \to o))$$
$$x_1 : (o \to o), x_2 : o \vdash_{\mathsf{s}} \lambda z^o.\, x_1 \equiv N_2 : (o \to (o \to o))$$

are denoted by isomorphic morphisms in the categorical model, but N_1 is unsafe whereas N_2 is safe. (This is because in N_1, the variable x has to be introduced first in the derivation tree, whereas in N_2, although x_1 needs to be introduced first, x_2 can be added to the context at the end of the derivation using the weakening rule.)

To avoid this kind of problem we have to define an alternative notion of safe lambda calculus with product. One way is to require that for every context-variable of type $A \times B$ the equality $\operatorname{ord} A = \operatorname{ord} B$ holds. Another solution is to forbid the use of variables of product type and only allow product types for terms created with the pairing rule. But these two approaches are rather restrictive. A better approach consists in changing the system to allow the formation of terms like N_2. This can be done by adding a new kind of weakening rule that alters the type of context-variables rather than adding new variables to the context:

$$(\mathsf{wk}^{\times})\ \frac{\Gamma, x : A \vdash_{\mathsf{s}} s : C}{\Gamma, x : A \times B \vdash_{\mathsf{s}} s\,[(\pi_1 x)/x] : C}\ .$$

Semantically, this rules is equivalent to the weakening rule because in the categorical model of the simply-typed lambda calculus, if s is denoted by a morphism $[\![s]\!] : \Gamma \times A \to C$ then $\Gamma, x : A \times B \vdash_{\mathsf{st}} s\,[(\pi_1 x)/x] : C$ and $\Gamma, x : A, y : B \vdash_{\mathsf{st}} s\,[(\pi_1 x)/x] : C$ are denoted by the morphisms $(id_\Gamma \times \pi_1^{A\times B}); [\![s]\!]$ and $\pi_1^{(\Gamma\times A)\times B}; [\![s]\!]$. These two denotations are the same since $id_\Gamma \times \pi_1^{A\times B} = \langle \pi_1^{\Gamma\times(A\times B)}; id_\Gamma, \pi_2^{\Gamma\times(A\times B)}; \pi_1^{A\times B}\rangle$, which by associativity of the product is isomorphic to $\langle \pi_1^{(\Gamma\times A)\times B}; \pi_1^{\Gamma\times A}, \pi_2^{(\Gamma\times A)\times B)}; \pi_2^{\Gamma\times A}\rangle = \pi_1^{(\Gamma\times A)\times B}$.

Example 6.4. With the addition of this rule to the system, both N_1 and N_2 are typable.

Again it is easy to see that the basic property of the safe lambda calculus still holds and therefore all the basic results showed in Chapter 3 also hold. Moreover, for every term typable with these rules there exists some term typable in safe $\Lambda^{\times}_{\to}$ with an isomorphic denotation (in the categorical model of the simply-typed lambda calculus).

6.1.2 Incremental closed category

We first recall some basic categorical notions and fix some notations before introducing the notion of incremental closed category.

Basic definitions

A ***category*** $\mathbf{C}$ is given by a class $\mathrm{Obj}(\mathbf{C})$ of objects and a class $\mathrm{Hom}(\mathbf{C})$ of morphisms between objects: for each pair of objects A, B, a set of morphisms $\mathbf{C}(A, B)$, written $f : A \to B$, where A is the domain and B is the codomain. Further for every three objects A, B and C, and morphisms $f : A \to B$ and $g : B \to C$ there is a composite morphism written $f; g$ or $g \circ f$ such that the composition operation is associative; and for each object A there is a morphism id_A that is the identity for composition.

Two objects A and B are said to be ***isomorphic***, written $A \cong B$, if there exists a pair of morphism $f : A \to B$ and $g : B \to A$ such that $f \circ g = id_B$ and $g \circ f = id_A$.

A ***subcategory*** of a category $\mathbf{C}$ is a category whose objects and morphisms are respectively objects and morphisms of $\mathbf{C}$. It is a *lluf* subcategory if it contains all the objects of $\mathbf{C}$.

An object I is ***terminal*** if for every object A there is a unique morphism ϵ_A from A to I.

A category has ***products*** if for every two objects A and B there is an object $A \times B$ and two morphisms π_1, π_2 mapping $A \times B$ to A and B respectively such that for every morphisms $f : C \to A$, $g : C \to B$, there is a unique morphism $\langle f, g\rangle : C \to A \times B$, called the ***pairing*** of f and g, such that $\pi_2 \circ \langle f, g\rangle = g$ and $\pi_1 \circ \langle f, g\rangle = f$.

A category has ***exponentials*** if for every two objects B and C there is an object C^B and a morphism $ev_{B,C} : (C^B \times B) \to C$ such that for every object A and morphism $f : (A \times B) \to C$ there is a unique morphism $\Lambda(f) : A \to C^B$ such that the following diagram commutes:

$$\begin{tikzcd}
A \times B \arrow[d, dashed, "\Lambda(f) \times id_B"'] \arrow[dr, "f"] & \\
C^B \times B \arrow[r, "ev_{B,C}"'] & C
\end{tikzcd}$$

Definition 6.5. A ***cartesian closed category***, CCC for short, is a category with a terminal object, binary products and exponentials.

In a CCC, the product is commutative and associative, further every finite (possibly empty) family of objects $\{A_j \mid j \in J\}$, for some finite index set I, has a product written $\Pi_{j\in J} A_j$. The empty product is given by the terminal object.

From now on when considering a CCC we use the notation I to refer to its terminal object, ϵ_A to denote the unique morphism $A \to I$, and π_i for the projection $A_1 \times A_2 \to A_i$, $i \in \{1, 2\}$ for some objects A_1, A_2. We say that a morphism is a ***weakener*** if it is either a projection or ϵ_A for some object A.

Incremental closed category

Definition 6.6. A ***pre-incremental closed category*** is a triple $(\mathbf{C}, \mathrm{ord}, \mathrm{dro})$ where $\mathbf{C}$ is a CCC and $\mathrm{ord} : \mathrm{Obj}(\mathbf{C}) \to \mathbb{N} \cup \{-1\}$ and $\mathrm{dro} : \mathrm{Obj}(\mathbf{C}) \to \mathbb{N} \cup \{\infty\}$ are functions satisfying the following conditions for all objects A, B:

(i) $A \cong B$ implies $\mathrm{ord}\, A = \mathrm{ord}\, B$ and $\mathrm{dro}\, A = \mathrm{dro}\, B$,

(ii) $\mathrm{ord}\, A = -1$ iff $\mathrm{dro}\, A = \infty$ iff $A \cong I$,

(iii) for $A, B \ncong I$, $\mathrm{ord}(A \times B) = \max(\mathrm{ord}\, A, \mathrm{ord}\, B)$ and $\mathrm{dro}(A \times B) = \min(\mathrm{dro}\, A, \mathrm{dro}\, B)$,

(iv) for $B \ncong I$, $\mathrm{ord}(B^A) = \max(1 + \mathrm{ord}\, A, \mathrm{ord}\, B)$ and $\mathrm{dro}(B^A) = \max(1 + \mathrm{ord}\, A, \mathrm{dro}\, B)$,

(v) for every object $A \not\cong I$ we have $\mathrm{ord}(A) \geq \mathrm{dro}(A)$.

(Observe that (i) implies $\mathrm{ord}(A \times I) = \mathrm{ord}(I \times A) = \mathrm{ord}(A^I) = \mathrm{ord}\, A$ for every object A and similarly for the function dro.)

We say that a morphism $f : A \to B$ is ***incremental*** if we have $\mathrm{dro}(A) \geq \mathrm{ord}(B)$. It is incremental ***modulo weakening*** if it can be written $f = \sigma_1; \ldots; \sigma_k; f'$ for some incremental morphism f' and weakeners $\sigma_1, \ldots, \sigma_k$, $k \geq 0$.

Composing two incremental morphisms gives an incremental morphism:

Lemma 6.7. *For every objects A, B and C of a pre-incremental closed category* $(\mathbf{C}, \mathrm{ord}, \mathrm{dro})$ *where $B \not\cong I$, if* $\mathrm{dro}(A) \geq \mathrm{ord}(B)$ *and* $\mathrm{dro}(B) \geq \mathrm{ord}(C)$ *then* $\mathrm{dro}(A) \geq \mathrm{ord}(C)$*.*

Proof. This follows from the fact that $\mathrm{ord}(B) \geq \mathrm{dro}(B)$ for $B \not\cong I$. □

Composing an incremental morphism with a weakener also gives an incremental morphism:

Lemma 6.8. *Let $f : A \to B$ be an incremental morphism and $\sigma : B \to C$ be a weakener then $f;\sigma$ is incremental.*

Proof. There are two cases: Suppose that $\sigma = \epsilon_B : B \to I$ then we have $\mathrm{dro}\, A \geq \mathrm{ord}\, B \geq \mathrm{ord}\, I = -1$. Otherwise σ is the projection $\pi_i : B_1 \times B_2 \to B_i$ for $i \in \{1, 2\}$ and $B = B_1 \times B_2$, in which case we have $\mathrm{dro}\, A \geq \mathrm{ord}(B_1 \times B_2) \geq \mathrm{ord}\, B_i$. □

Definition 6.9 (Incremental closed categories)**.** An ***incremental closed category***, ICC for short, is a 4-tuple $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$ such that $(\mathbf{C}, \mathrm{ord}, \mathrm{dro})$ is a pre-incremental closed category and $\mathbf{I}$ is a lluf subcategory of $\mathbf{C}$ such that:

1. the terminal object of $\mathbf{C}$ is also terminal in $\mathbf{I}$ (so in particular it has the morphism $\epsilon_A : A \to I$ for all object A);
2. it preserves the products of $\mathbf{C}$:
   - it contains all the projections: for all objects C_1 and C_2, $\pi_1 : C_1 \times C_2 \to C_1$ and $\pi_2 : C_1 \times C_2 \to C_2$ are in $\mathrm{Hom}(\mathbf{I})$;
   - it is closed under pairing: if $f : C \to A$ and $g : C \to B$ are in $\mathrm{Hom}(\mathbf{I})$ then so is $\langle f, g \rangle$;
3. it has *incremental* exponentials:
   - it is closed under *incremental* currying: if $f : (A \times B) \to C \in \mathrm{Hom}(\mathbf{I})$ with $\mathrm{dro}\, A \geq \mathrm{ord}(C^B)$ then $\Lambda(f) : A \to C^B \in \mathrm{Hom}(\mathbf{I})$;
   - it is closed under right composition with evaluation maps followed by *incremental* currying: if $f : A_1 \times A_2 \to C^B \times B \in \mathrm{Hom}(\mathbf{I})$ and $\mathrm{dro}\, A_1 \geq \mathrm{ord}(C^{A_2})$ then $\Lambda(f; ev_{B,C}) : A_1 \to C^{A_2} \in \mathrm{Hom}(\mathbf{I})$.

(Note that the evaluation maps of the ambient CCC are not required to belong to the ICC.)

Let $(\mathbf{C}, \mathrm{ord}, \mathrm{dro})$ be a pre-incremental closed category. We define its ***canonical ICC*** as $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$ where $\mathbf{I}$ is the lluf subcategory obtained by keeping just the morphisms that are incremental *modulo weakening*. Formally for every objects A, B:

$$\begin{aligned}\mathbf{I}(A, B) = \{f \in \mathbf{C}(A, B) \mid f = \sigma_1; \ldots; \sigma_k; f', &\text{ for some object } A' \text{ with } \mathrm{dro}\, A' \geq \mathrm{ord}\, B, \\ &\text{morphism } f' \in \mathbf{C}(A', B) \text{ and weakeners } \sigma_1, \ldots, \sigma_k, k \geq 0\}\ .\end{aligned}$$

In particular if $\mathrm{dro}\, A \geq \mathrm{ord}\, B$ then $\mathbf{I}(A, B) = \mathbf{C}(A, B)$.

Proposition 6.10. *Let* $(\mathbf{C}, \mathrm{ord}, \mathrm{dro})$ *be a pre-incremental closed category. Then its canonical ICC* $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$ *is an ICC.*

Proof. We first show that $\mathbf{I}$ is a lluf subcategory of $\mathbf{C}$: The identity morphisms are all incremental therefore they are in $\mathrm{Hom}(\mathbf{I})$. We now show closure under composition. Take two morphisms $f : A \rightarrow B$ and $g : B \rightarrow C$. We have $f = \sigma_1; \ldots \sigma_k; f'$ and $g = \rho_1; \ldots \rho_l; g'$ for some weakeners σ_i, ρ_j, $1 \leq i \leq k$, $1 \leq j \leq l$, $k, l \geq 0$ and incremental morphisms f' and g'. We do a case analysis on $(k, l) \in \mathbb{N} \times \mathbb{N}$.

(a) Suppose $k = l = 0$: If $B \ncong I$ then by Lemma 6.7, $f; g$ is incremental; otherwise since I is the terminal object, f is necessarily the the unique morphism $\epsilon_A : A \rightarrow I$. Thus since ϵ_A is a weakener $f; g = \epsilon_A; g$ belongs to $\mathbf{I}$.

(b) Suppose $k, l \geq 0$. By associativity we have $f; g = \sigma_1; \ldots \sigma_k; h$ where $h = (f'; \rho_1; \ldots \rho_l); g'$. Applying Lemma 6.8 l times shows that the morphism $f'; \rho_1; \ldots \rho_l$ is incremental, thus by (a) we have that h belongs to $\mathbf{I}$: $h = \tau_1; \ldots \tau_n; h'$ for some weakeners $\tau_1 \ldots \tau_n$, $n \geq 0$. Hence $f; g = \sigma_1; \ldots \sigma_k; \tau_1; \ldots \tau_n; h'$ also belongs to $\mathbf{I}$.

Hence $\mathbf{I}$ is a lluf subcategory. Further I is a terminal object (because $\mathbf{I}(A, I) = \mathbf{C}(A, I)$); it contains the projections (A projection $\pi_i : C_1 \times C_2 \rightarrow C_1$ that is not incremental can always be written $\pi_i = \pi_i; id_{C_i}$ where $id_{C_i} : C_i \rightarrow C_i$ denotes the identity morphism which is incremental.). Finally since it contains all the incremental morphisms, it is closed under pairing, incremental currying, and right composition with the evaluation maps followed by incremental currying. Hence $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$ is an ICC. $\square$

Remark 6.11 (Homogeneous incremental closed category) It is also possible to interpret type homogeneity (see Sec. 2.2.2) categorically. A non-terminal object A of a pre-incremental closed category $(\mathbf{C}, \mathrm{ord}, \mathrm{dro})$ is said to be ***homogeneous*** if

- A is a base object (neither a product nor an exponential);
- or $A = B \times C$ where B and C are homogeneous and $\mathrm{ord}\, B \geq \mathrm{ord}\, C$;
- or $A = B \rightarrow C$ where B and C are homogeneous and $\mathrm{ord}\, B \geq \mathrm{ord}\, C - 1$.

A sub-category of an ICC consisting of the homogeneous objects plus the terminal object I, and incremental morphisms (but not those that are incremental only *modulo weakening*) is then called an ***homogeneous incremental closed category***.

Order-enrichment

In order to model applied lambda calculi with recursion, one needs to impose further requirement on the category. The condition called *rationality* [AM99] is sufficient for a CCC to interpret PCF. We reproduce the definition here: A *pointed poset* is a partially ordered set with a least element. A category $\mathbf{C}$ is ***pointed-poset*** enriched (ppo-enriched) if

- every hom-set has a pointed poset structure $(\mathbf{C}(A, B), \sqsubseteq_{A,B}, \perp_{A,B})$;
- composition, pairing and currying are monotone;
- composition is *left-strict*: for all $f : A \rightarrow B$, $\perp_{B,C} \circ f = \perp_{A,C}$.

A category $\mathbf{C}$ is ***rational*** if it is ppo-enriched and for all $f : A \times B \rightarrow B$, the chain defined by $f^{(0)} = \perp_{A,B}$, $f^{(k+1)} = f \circ \langle id_A, f^{(k)} \rangle$ has a least upper bound denoted by f^{∇} such that for all $g : C \rightarrow A$, $h : B \rightarrow D$, $g \circ f^{\nabla} \circ h = \bigcup_{k \in \omega} g \circ f^{(k)} \circ h$.

We now extend this definition to ICCs as follows:

Definition 6.12. An ICC $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$ is ***rational*** if $\mathbf{C}$ is rational and $\mathbf{I}$ is complete with respect to $(\cdot)^{\nabla}$ (*i.e.*, if $f : A \times B \rightarrow B$ is a morphism of $\mathbf{I}$ then so is f^{∇}).

6.1.3 Categorical semantics

Consider a typed lambda calculus extended with a set of constants and function symbols together with a set of reduction rules giving the operational interpretation of these functions. A ***model*** of a typed lambda calculus in a cartesian closed category is specified by giving:

- For every ground type T an object $[\![T]\!]$ of the category. This suffices to interpret any simple type T as an object $[\![T]\!]$ using products and exponentials;

- for every constant k of type T a morphism $[\![K]\!]$ of type $[\![T]\!]$;
- for every function symbol f of type $A_1 \times \cdots \times A_n \to B$, a morphism $[\![f]\!]$ of type $[\![A_1]\!] \times \cdots \times [\![A_n]\!] \to [\![B]\!]$.

It is then possible to specify the interpretation of any term-in-context $\Gamma \vdash M : T$ by induction on the structure of the term [Cro93]. The ***model*** is said to be *sound* if whenever M reduces to N using the small-step semantics of the language then M and N have the same denotation in the model.

Proposition 6.13 (Models of safe typed lambda calculi)**.** *Let $\mathcal{L}$ be a typed lambda calculus and* $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$ *be an ICC. If* $\mathbf{C}$ *provides a sound model of* $\mathcal{L}$ *then* $\mathbf{I}$ *provides a sound model of the safe fragment of* $\mathcal{L}$.

Proof. The interpretation $[\![\cdot]\!]$ of the safe lambda calculus with product in $\mathbf{I}$ is induced by the standard interpretation in the CCC: Ground types are interpreted as objects of the category, this suffices to interpret any simple type T as an object $[\![T]\!]$ using products and exponentials. A term-in-context $\Gamma \vdash M : T$ is then interpreted by a morphism $[\![\Gamma]\!] \to [\![T]\!]$.

We show by induction on the derivation tree of a safe term-in-context $\Gamma \vdash_{\mathsf{s}} M : T$ that its denotation $[\![\Gamma \vdash_{\mathsf{s}} M : T]\!]_{\mathbf{C}}$ in $\mathbf{C}$ is also a morphism of the subcategory $\mathbf{I}$. The (var) axiom is interpreted by the identity morphisms which all belong to the ICC. The case $(\times)$, (π_1) and (π_2) follow by the fact that an ICC contain projections and is closed by pairing. The weakening rule (wk) is interpreted by composition with the projection morphisms. For the rule (app), we have $[\![\Gamma \vdash_{\mathsf{s}} N_0\, N_1 \ldots N_n : B]\!] = \langle [\![N_0]\!], [\![N_1]\!], \ldots, [\![N_n]\!] \rangle \fatsemi ev_{(A_1 \times \ldots \times A_n), B}$ where $\mathrm{dro}\, \Gamma \geq \mathrm{ord}\, B$. We can thus conclude using the I.H. and the fact that an ICC is closed by right composition with evaluation followed by incremental currying (here it is the dummy currying of a morphism $\Gamma \times I \to B$). Rule (abs): Let $f : \Gamma \times (A_1 \times \ldots \times A_n) \to T$ be the denotation of the premise. The conclusion of the rule is denoted by the curried morphism $\Lambda(f) : \Gamma \to T^{(A_1 \times \ldots \times A_n)}$ which, by the side-condition of the rule, is incremental. If the premise is not an application then we conclude using the fact that an ICC is closed by incremental currying. Otherwise we conclude using the fact that an ICC is closed by composition with evaluation followed by incremental currying.

Hence for every safe term M, we can define its interpretation $[\![M]\!]_{\mathbf{I}}$ in $\mathbf{I}$ to be its interpretation in $\mathbf{C}$: $[\![M]\!]_{\mathbf{I}} \stackrel{\mathrm{def}}{=} [\![M]\!]_{\mathbf{C}}$. The soundness of the ICC model follows from that of the CCC model. ☐

Example 6.14 (Model of safe PCF)**.** It is a well-known fact that any rational CCC in which we have fixed an interpretation for base types, PCF constants and function symbols provides a sound model of PCF [AMJ94]. Therefore any rational ICC provides a sound model of safe PCF. The interpretation of safe PCF in the ICC coincides with its interpretation in the ambient pre-incremental closed category [AMJ94]: Each constant and first-order function of PCF of type T is interpreted by some morphism $c : I \to [\![T]\!]$, and because the category is rational, the Y combinator Y_A for every object A can be interpreted by the morphism $\Theta_A^{\triangledown} : I \to A^{A^A}$ where

$$\Theta_A = [\![F : (A \to A) \to A \vdash \lambda f^{A \to A}.f(Ff) : (A \to A) \to A]\!] \ .$$

6.1.4 Quotiented category

Let $\mathbf{C}$ be a rational CCC. A *precongruence* $\lesssim$ on $\mathbf{C}$ is defined as a family of preorders $\lesssim_{A,B} \subseteq \mathbf{C}(A,B) \times \mathbf{C}(A,B)$ such that $\sqsubseteq_{A,B} \subseteq \lesssim_{A,B}$, composition, pairing, currying are $\lesssim$-monotonous, and the preorders satisfy some continuity property [AMJ94]. Given a precongruence, the quotiented category $\mathbf{C}/\lesssim$ is defined as follows: the objects are those of $\mathbf{C}$, and a morphism in $\mathbf{C}/\lesssim (A,B)$ is an equivalence class $[f]$ of $\mathbf{C}(A,B)$ modulo the equivalence relation induced by $\lesssim_{A,B}$. A partial ordering $\leq_{A,B}$ on $\mathbf{C}/\lesssim (A,B)$ can then be defined as follows:

$$[f] \leq_{A,B} [g] \iff f \lesssim_{A,B} f \ .$$

Lemma 6.15 ([AMJ94])**.** *If* $\lesssim$ *is a precongruence on a rational CCC* $\mathbf{C}$ *then* $\mathbf{C}/\lesssim$ *is a rational CCC.*

The notion of quotient category extends naturally to ICCs: the precongruence $\lesssim$ on $\mathbf{I}$ for some ICC $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$, is defined similarly as CCC precongruences except that monotonicity is required for *incremental* currying only. This naturally gives rise to the notion of quotiented category $\mathbf{I}/\lesssim$.

Lemma 6.16. *Let* $(\mathbf{C}, I, \mathrm{ord}, \mathrm{dro})$ *be an ICC, and let* $\lesssim$ *be a precongruence on* $\mathbf{C}$*. Then:*

(i) $(\mathbf{C}/\lesssim, \mathbf{I}/\lesssim, \mathrm{ord}, \mathrm{dro})$ *is an ICC;*

(ii) If $(\mathbf{C}, I, \mathrm{ord}, \mathrm{dro})$ *is rational then so is* $(\mathbf{C}/\lesssim, \mathbf{I}/\lesssim, \mathrm{ord}, \mathrm{dro})$.

Proof. (i) Since $\lesssim$ is a CCC precongruence, it is in particular an ICC precongruence therefore the quotiented category $\mathbf{I}/\lesssim$ is well-defined. Since $\mathbf{I}$ is a subcategory of $\mathbf{C}$, each equivalent class of morphisms of $\mathbf{I}$ is a subset of some equivalent class of morphisms of $\mathbf{C}$; therefore, up to an obvious isomorphism, the category $\mathbf{I}/\lesssim$ is a lluf subcategory of $\mathbf{C}/\lesssim$. Finally, the incremental closure of $\mathbf{I}$ immediately implies that of $\mathbf{I}/\lesssim$.

(ii) Suppose $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$ is rational. By definition this means that $\mathbf{C}$ is rational and $\mathbf{I}$ is complete with respect to the operation $(\cdot)^\triangledown$. By Lemma 6.15, $\mathcal{C}/\lesssim$ is also a rational CCC, therefore by (i), $\mathbf{I}/\lesssim$ is a lluf subcategory of a rational CCC.

Let $[f] : A \times B \to B$ be an equivalence class morphism in $\mathbf{I}/\lesssim$. It is also a morphism of the category $\mathbf{C}/\lesssim$, therefore by CCC rationality the least upper bound of the chain $[f]^{(n)}$ is given by $[f^\triangledown]$ [AMJ94]. Since $\mathbf{I}$ is $(\cdot)^\triangledown$-complete this implies that $[f^\triangledown]$ is also in $\mathbf{I}/\lesssim$. Thus $\mathbf{I}/\lesssim$ is also $(\cdot)^\triangledown$-complete.

Hence $(\mathbf{C}/\lesssim, \mathbf{I}/\lesssim, \mathrm{ord}, \mathrm{dro})$ is a rational ICC. ∎

6.1.5 The internal language of incremental closed categories

By a well-known result by Lambek, the simply-typed lambda calculus is the language of cartesian closed categories [Lam86]: For every cartesian closed category $\mathbf{C}$ one can construct a typed lambda calculus $L(\mathbf{C})$ called the ***internal language*** of the CCC. And for every typed lambda calculus $\mathcal{L}$ we can construct a CCC $Cl(\mathcal{L})$ that soundly interprets $\mathcal{L}$; this category is called the ***CCC generated*** by $\mathcal{L}$ or also the ***canonical classifying category*** of $\mathcal{L}$ [Cro93]. Furthermore these two transformations establish an equivalence of categories which means that their composites are naturally isomorphic to the identity functors:

$$\mathbf{C} \cong Cl(L(\mathbf{C})), \qquad \mathcal{L} \cong L(Cl(\mathcal{L})) \ . \tag{6.1}$$

Does a similar correspondence hold between ICCs and safe typed lambda calculi? Following [Lam86], it is possible to adapt the notion of *internal language* to ICCs. Given an ICC $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$, we can define its ***internal language*** $L(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$ as the typed lambda calculus whose types are the objects of $\mathbf{I}$, and terms of type A are freely generated from the basic constants (given by arrows $a : I \to A$) and variable $x : A$ (given by indeterminate morphisms $x : I \to A$) by the term forming operations induced by the maps of $\mathbf{I}$ (pairing, incremental currying, composition with projection, and composition with evaluation): the formation rules are the same as those of the internal language of the ambient CCC except that the abstraction and application rules have a side-condition ensuring that the context variables have order greater than the order of the term being formed. This does not allow the formation of *almost-safe* terms, this language is thus precisely the *long-safe* fragment of the internal language of $\mathbf{C}$:

Definition 6.17. The ***internal language*** of an ICC $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$ is defined as

$$L(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro}) \stackrel{\mathrm{def}}{=} \text{long-safe}_{\widetilde{\mathrm{ord}}}(L(\mathbf{C}))$$

where

- for every typed lambda calculus $\mathcal{L}$ and function $f : \mathbb{T} \rightarrow \mathbb{N}$, long-safe$_f(\mathcal{L})$ denotes the long-safe fragment of $\mathcal{L}$ (Def. 3.31) where the side-condition in the application and abstraction rules is defined using the type-order function f;
- the type-order function $\widetilde{\mathrm{ord}} : \mathbb{T} \rightarrow \mathbb{N}$ is defined as follows: for every type $T \in \mathbb{T}$, $\widetilde{\mathrm{ord}}\, T = \mathrm{ord}\, [\![T]\!]$, where $[\![T]\!]$ is the denotation of the type T in the model $\mathbf{C}$ of $L(\mathbf{C})$.

Definition 6.18. Let $\mathcal{L}$ be a typed lambda calculus over simple types $\mathbb{T}$ and $\mathrm{ord} : \mathbb{T} \rightarrow \mathbb{N}$ be some ranking function on types. We define the functions $\mathrm{ord}^+ : \mathrm{Obj}(Cl(\mathcal{L})) \rightarrow \mathbb{N} \cup \{-1\}$ and $\mathrm{ord}^- : \mathrm{Obj}(Cl(\mathcal{L})) \rightarrow \mathbb{N} \cup \{\infty\}$ on the objects of the category $Cl(\mathcal{L})$ as follows:

$$\begin{array}{lll} \mathrm{ord}^+(A) = \mathrm{ord}\, T, & \mathrm{ord}^-(A) = \mathrm{dro}\, T, & \text{if } A = [\![T]\!] \text{ for some type } T \in \mathbb{T}, \\ \mathrm{ord}^+(A) = -1, & \mathrm{ord}^-(A) = \infty, & \text{if } A \cong I, \\ \mathrm{ord}^+(A) = 0, & \mathrm{ord}^-(A) = 0, & \text{otherwise.} \end{array}$$

where the function $\mathrm{dro} : \mathbb{T} \rightarrow \mathbb{N}$ is defined as: $\mathrm{dro}(T) = \mathrm{ord}(T)$ for every base type T, $\mathrm{dro}(T_1 \times T_2) = \min(\mathrm{dro}\, T_1, \mathrm{dro}\, T_2)$ and $\mathrm{dro}(T_1 \rightarrow T_2) = \max(1 + \mathrm{ord}\, T_1, \mathrm{dro}\, T_2)$ for every types $T_1, T_2 \in \mathbb{T}$.

(These two functions are well-defined because in $Cl(\mathcal{L})$, for every type $T \in \mathbb{T}$ we have $[\![T]\!] \ncong I$ and for every simple types T_1, T_2, $T_1 \neq T_2$ implies $[\![T_1]\!] \neq [\![T_2]\!]$.)

A *type-order function* is a function $\mathrm{ord} : \mathbb{T} \rightarrow \mathbb{N}$ satisfying $\mathrm{ord}(T_1 \times T_2) = \max(\mathrm{ord}\, T_1, \mathrm{ord}\, T_2)$ and $\mathrm{ord}(T_1 \rightarrow T_2) = \max(1 + \mathrm{ord}\, T_1, \mathrm{ord}\, T_2)$ for every types $T_1, T_2 \in \mathbb{T}$. Clearly, for every such function, the triple $(\mathbf{C}, \mathrm{ord}^+, \mathrm{dro}^-)$ defines a pre-incremental closed category (Def. 6.6).

Definition 6.19. The ***canonical classifying ICC*** of (or ***ICC generated*** by) $\mathcal{L}$ with respect to a type-order function ord, written $ICl_{\mathrm{ord}}(\mathcal{L})$, is defined as the canonical ICC induced by the pre-ICC $(Cl(\mathcal{L}), \mathrm{ord}^+, \mathrm{dro}^-)$:

$$ICl_{\mathrm{ord}}(\mathcal{L}) \stackrel{\mathrm{def}}{=} (Cl(\mathcal{L}), \mathbf{I}, \mathrm{ord}^+, \mathrm{ord}^-)$$

where $\mathbf{I}$ denotes the canonical ICC of $(Cl(\mathcal{L}), \mathrm{ord}^+, \mathrm{dro}^-)$.

Proposition 6.20.

(i) For every typed lambda calculus $\mathcal{L}$ with simple types $\mathbb{T}$ and type-order function $\mathrm{ord} : \mathbb{T} \rightarrow \mathbb{N}$ *we have:*

$$L(ICl_{\mathrm{ord}}(\mathcal{L})) \cong \textit{long-safe}_{\mathrm{ord}}(\mathcal{L}) \ .$$

(ii) For every pre-incremental closed category $(\mathbf{C}, \mathrm{ord}, \mathrm{dro})$ *with canonical ICC* $\mathbf{I}$ *we have:*

$$ICl_{\widetilde{\mathrm{ord}}}(L(\mathbf{C})) \cong (\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro}) \ .$$

Proof. This is an immediate consequence of (6.1) and definitions 6.17 and 6.19. (i) follows from the fact that $\widetilde{\mathrm{ord}^+} = \mathrm{ord}$. (ii) follows from the fact that $\widetilde{\mathrm{ord}}^+ = \mathrm{ord}$ and $\widetilde{\mathrm{ord}}^- = \mathrm{dro}$. ☐

Intrinsically safe fragment Let $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$ be an ICC. We define the *intrinsically safe fragment* $LI(\mathbf{I})$ of $L(\mathbf{C})$ as the language consisting of the terms whose denotations in $\mathbf{C} \cong Cl(L(\mathbf{C}))$ are also in $\mathbf{I}$:

$$LI(\mathbf{I}) \stackrel{\mathrm{def}}{=} \{\ t \in L(\mathbf{C}) \mid [\![t]\!] \in \mathrm{Hom}(\mathbf{I})\} \ .$$

This definition implies $[\![LI(\mathbf{I})]\!] = \mathbf{I}$. This language satisfies the basic property of the safe lambda calculus:

Lemma 6.21. *Let* $(\mathbf{C}, \mathbf{I}, \mathrm{ord}, \mathrm{dro})$ *be an ICC. For every term M of $LI(\mathbf{I})$, the free variables of M have order greater than* $\mathrm{ord}\, M$*.*

Proof. Lambek [Lam86] defines a functor $[\![\cdot]\!] : \mathcal{L} \to \mathbf{C}$ such that every term M of the language $\mathcal{L}$ of type B with free variables of type $A_1, \ldots, A_n$ is denoted by a morphism in $\mathbf{C}(A_1 \times \ldots \times A_n, B)$. Take $\mathcal{L}$ to be $LI(\mathbf{I})$, then by definition M is denoted by an incremental morphism therefore $\operatorname{dro}(A_1 \times \ldots \times A_n) \geq \operatorname{ord} B$. We then have for $1 \leq i \leq n$:

$$\operatorname{ord} A_i \geq \operatorname{dro} A_i \geq \operatorname{dro}(A_1 \times \ldots \times A_n) \geq \operatorname{ord} B \ . \qquad \square$$

The language $LI(\mathbf{I})$, however, is *not* the safe fragment of the internal language of $\mathbf{C}$. Indeed, since safety is only preserved by β-reduction but not by β-equality, it is possible to have an unsafe term U in $L(\mathbf{C})$ with a safe beta-nf $\beta_{\mathsf{nf}}(U)$; since $\beta_{\mathsf{nf}}(U)$ is safe, its denotation is an incremental morphism and therefore it belongs to $LI(\mathbf{I})$, but by soundness of the model $\mathbf{C}$, the terms U and $\beta_{\mathsf{nf}}(U)$ have the same denotation, hence the unsafe term U must also belong to $LI(\mathbf{I})$.

6.2 The game model

Our aim for the rest of this chapter is to construct a category of games that is incremental closed, thus giving rise to a game model of the safe lambda calculus. We start by introducing the class of *closed P-incremental justified strategies* and then show that it is closed under composition. This then allows us to construct an ICC category with game as objects and closed P-incremental justified strategies as morphisms.

We make the following assumptions on games. Let $\perp$ denote the game whose arena has a single initial question move and no answers. For every game $A \neq \perp$:

(A1) Each question move in the game enables at least one answer move;

(A2) Answer moves do not enable any other move.

Clearly, PCF and IA games all satisfy these two assumptions.

A game is said to be ***prime*** if it has a single initial move; a type is *prime* if its game denotation is prime.

6.2.1 Order of a move

We recall the definition of a move-order (Def 2.74). Let $A = \langle M, \lambda, \vdash \rangle$ be a game. We call $\vdash$-chain, any sequence of enabling moves $m_1 \vdash m_2 \vdash \ldots \vdash m_h$ where $h \in \mathbb{N}$ is called the *length* of the chain. The ***order of a question move*** q in A, written $\operatorname{ord}_A q$ (or just $\operatorname{ord} q$ where there is no ambiguity) is defined as the length of the longest $\vdash$-chain of questions starting from q minus 1. The order of an answer-move is defined as -1. (Alternatively, under assumptions (A1) and (A2), if $A \neq \perp$, the order of a (question or answer) move m is given by the length of the longest $\vdash$-chain starting from m minus 2.) The ***order of a game*** is defined as the maximal order of its (initial) moves: $\operatorname{ord} A = \max_{m \in M} \operatorname{ord}_A m$. The ***level*** of a move m, written $\operatorname{level}_A m$, is the length of the longest $\vdash$-chain ending with m. It is easy to see that the following relation holds for every question move q of a game $A \neq \perp$:

$$\operatorname*{ord}_A q + \operatorname*{level}_A q \leq \operatorname{ord} A \ .$$

Thus a move m is a question if and only if $\operatorname{ord} m \geq 0$, and it is an answer if and only if $\operatorname{ord} m = -1$.

We recall that for every type T built up from base types, product and function space, the order of T, written $\operatorname{ord} T$, is defined by induction as follows: A base type has order 0, $\operatorname{ord}(A \to B) = \max(1 + \operatorname{ord} A, \operatorname{ord} B)$, and $\operatorname{ord}(A \times B) = \max(\operatorname{ord} A, \operatorname{ord} B)$ for every types A and B. Clearly, this definition coincides with the definition given above: the order of a type is the order of the arena denoting it (*i.e.*, $\operatorname{ord} T = \operatorname{ord} [\![T]\!]$ for every type T).

Move-order after composition

Consider the game $X \multimap Y$ and let m be a move of $X \multimap Y$. We write $\mathrm{ord}_{X \multimap Y}\, m$ to denote the order of m in the game $X \multimap Y$. If m belongs to X (resp. Y) then we write $\mathrm{ord}_X\, m$ (resp. $\mathrm{ord}_Y\, m$) to denote the order of the move m in the game X (resp. Y).

Lemma 6.22. *Let A, B and C be three games. We have:*

$$\begin{array}{lllll}
\forall m \in A : & \mathrm{ord}_{A\multimap B}\, m = \mathrm{ord}_{A\multimap C}\, m \ , & & \\
\forall m \in B : & \mathrm{ord}_{A\multimap B}\, m \geq \mathrm{ord}_{B\multimap C}\, m & & \textit{for } m \textit{ initial,} \\
 & \mathrm{ord}_{A\multimap B}\, m = \mathrm{ord}_{B\multimap C}\, m & & \textit{for } m \textit{ non initial,} \\
\forall m \in C : & \mathrm{ord}_{A\multimap C}\, m \geq \mathrm{ord}_{B\multimap C}\, m & \iff \mathrm{ord}\, A \geq \mathrm{ord}\, B & \textit{for } m \textit{ initial,} \\
 & \mathrm{ord}_{A\multimap C}\, m = \mathrm{ord}_{B\multimap C}\, m & & \textit{for } m \textit{ non initial.}
\end{array}$$

The proof is immediate.

6.2.2 Well-bracketing

We call ***pending question*** of a sequence of moves $s \in L_A$ the last unanswered question in s.

Definition 6.23. A strategy σ is said to be ***P-well-bracketed*** if for every play $s\, a \in \sigma$ where a is a P-answer, a points to the pending question in s.

P-well-bracketing can be restated differently as the following proposition shows:

Proposition 6.24. *We make assumption (A1) and (A2). Let σ be a strategy on a game A. The following statements are equivalent:*

(i) σ is P-well-bracketed,

(ii) for $s\, a \in \sigma$ with a a P-answer, a points to the pending question in $\ulcorner s \urcorner$,

(iii) for $s\, a \in \sigma$ with a a P-answer, a points to the last O-question in $\ulcorner s \urcorner$,

(iv) for $s\, a \in \sigma$ with a a P-answer, a points to the last O-move in $\ulcorner s \urcorner$ with order $> \mathrm{ord}\, a$.

Proof. The result holds trivially if $A = \bot$ (the game with one initial question and no answers). Othwerise:

$(i) \iff (ii)$: [McC96a, Lemma 2.1] states that if P is to move then the pending question in s is the same as that of $\ulcorner s \urcorner$.

$(ii) \iff (iii)$: Assumption (A2) implies that the pending question in $\ulcorner s \urcorner$ is also the last O-question occurring in $\ulcorner s \urcorner$.

$(iii) \iff (iv)$: Because of assumption (A1) and (A2), for every move m, we have m is a question move if and only if $\mathrm{ord}\, m \geq 0$ if and only if $\mathrm{ord}\, m > \mathrm{ord}\, a = -1$. ☐

Lemma 6.25. *Under assumption (A2), if s is a justified sequence of moves satisfying alternation and visibility then any O-move (resp. P-move) in s points to an* unanswered *P question (resp. O-question).*

Proof. Suppose that an O-move c points to a P-move d that has already been answered by the O-move a. The sequence s as the following form:

$$s = \ldots d \ldots a \ldots c \ .$$

By O-visibility, d must belong to $\llcorner s_{<c} \lrcorner$. But since a is an answer, by assumption (A2), it cannot justify any P-move, therefore $\llcorner s_{<q} \lrcorner$ must contain an OP-arc "hoping" over a. We name the nodes of this arc d^1 and c^1:

$$s = \ldots d \ldots d^1 \ldots a \ldots c^1 \ldots c \ .$$

By P-visibility, d^1 must belong to $\ulcorner s_{<c^1} \urcorner$. Consequently, a does not belong to $\ulcorner s_{<c^1} \urcorner$ (otherwise the PO-arc $d \cdots a$ would cause the P-view to jump over d^1). Therefore there must be a PO-arc $d^2 \cdots c^2$ in $\ulcorner s_{<c^1} \urcorner$ hoping over a:

$$s = \ldots d \ldots d^1 \ldots c^2 \ldots a \ldots d^2 \ldots c^1 \ldots c \ .$$

This process can be repeated infinitely often by using alternatively O-visibility and P-visibility. This gives a contradiction since the sequence of moves $s_{<c}$ has finite length. Hence d cannot point to a question that has already been answered. Since by (A2) a question is enabled by another question, d is necessarily justified by an unanswered question. ☐

Lemma 6.26. *Under assumption (A2), if s is a P-well-bracketed justified sequence of moves of odd length satisfying alternation and visibility then all O-questions occurring in $\ulcorner s \urcorner$ are unanswered in s.*

Proof. We proof the first part by induction on s. The base case ($s = q$ with q initial O-move) is trivial. Suppose $s = s' \cdot q \cdot u \cdot m$. We have $\ulcorner s \urcorner = \ulcorner s' \urcorner \cdot q \cdot m$. Clearly m is unanswered in s. Let r be an O-question in $\ulcorner s' \urcorner$ and suppose that r is answered in s by some move a. By the induction hypothesis, r is unanswered in s' therefore a necessarily appears in the segment u:

$$s = \underbrace{\cdots\ r^O \cdots}_{s'}\ q^P\ \underbrace{\cdots\ a^P \cdots}_{u}\ m^O \quad .$$

But since m is justified by q, by Lemma 6.25 q must be unanswered in $s_{<m}$. In particular, the pending question at $s_{\leqslant a}$ cannot be r since the unanswered question q is played after r. This gives a contradiction since by well-bracketing a should answer the pending question. Hence r is unanswered in s. ☐

6.2.3 P-incremental justification

Recall the definition of P-incremental justification from Def. 5.1: A play sm of even length is said to be ***P-incrementally justified***, or *P-i.j.* for short, if m points to the last unanswered O-question in $\ulcorner s \urcorner$ with order strictly greater than $\operatorname{ord} m$. A strategy σ is said to be ***P-incrementally justified***, if all plays in σ ending with a P-question are P-incrementally justified.

Given a strategy σ, we will write $\mathcal{P}(\sigma)$ to denote the subset of σ consisting of plays whose even-length prefixes are all P-i.j. Hence P-i.j. strategies are precisely those satisfying the relation $\sigma = \mathcal{P}(\sigma)$.

Proposition 6.27. *Let σ be a* P-well-bracketed *strategy on a game A. Under assumptions (A1) and (A2), the following statements are equivalent:*

(i) σ is P-incrementally justified,

(ii) for $s\,q \in \sigma$ with q a P-question, q points to the last O-question in $\ulcorner s \urcorner$ with order $> \operatorname{ord} q$,

(iii) for $s\,q \in \sigma$ with q a P-question, q points to the last O-move in $\ulcorner s \urcorner$ with order $> \operatorname{ord} q$.

Proof. The result holds trivially if $A = \bot$. Otherwise: (i) iff (ii): By Lemma 6.26, O-questions occurring in $\ulcorner s \urcorner$ are all unanswered. (ii) iff (iii): By (A1), $\operatorname{ord} q \geq 0$ and by (A2), answer moves have order 0 therefore answer moves all have order $\leq \operatorname{ord} q$. ☐

P-incremental justification can be viewed as a generalization of well-bracketing to question moves: the third statement in the above proposition is the counterpart of Proposition 6.24(vi). Putting propositions 6.27 and 6.24 together we obtain:

Proposition 6.28. *Under assumption (A1) and (A2), a strategy σ is* P-well-bracketed *and* P-incrementally justified *if and only if for $s\,m \in \sigma$, m points to the last O-move in $\ulcorner s \urcorner$ with order $> \operatorname{ord} m$.*

6.2.4 Closed P-incremental justification

Definition 6.29. An even-length play sm on some game $A \rightarrow B$ is said to be ***closed P-incrementally justified*** (closed P-i.j. for short) just if

(i) sm is P-incrementally justified;

(ii) and if m is an initial move in A then its justifier n (initial in B) satisfies $\operatorname{ord}_A m \geq \operatorname{ord}_B n$.

A strategy σ is ***closed P-i.j.*** just if all plays in σ ending with a P-questions are closed P-i.j.

Example 6.30. For every game A, the identity strategy id_A is closed P-i.j.

Lemma 6.31. *Let $\sigma : A \multimap B$ be a P-i.j. strategy.*

(i) If for each initial move m of A occurring in some play of σ we have $\operatorname{ord}_A m \geq \operatorname{ord} B$, then σ is closed P-i.j.

(ii) Suppose that $A = A_1 \times \ldots \times A_n$ where each A_i is a prime arena for $i \in \{1..n\}$. If every initial move m_i of A_i that occurs in some play of σ we have $\operatorname{ord} A_i \geq \operatorname{ord} B$ then σ is closed P-i.j.

Proof. (i) This is a direct consequence of the definition since $\operatorname{ord} B \geq \operatorname{ord}_B b$ for every move b initial in B. (ii) Take an initial move m of A. We have $\operatorname{ord}_A m = \operatorname{ord}_{A_i} m$ for some i. This is in turn equal to $\operatorname{ord} A_i$ since A_i is prime. By hypothesis it is greater than $\operatorname{ord} B$ hence we can conclude using (i). ☐

Example 6.32. The simply-typed term $x : (o^1 \rightarrow o^2) \times o^3 \vdash_{\mathsf{st}} \lambda y^o.\pi_2 x : o^4 \rightarrow o^5$ has a P-i.j. denotation. The second part of the previous Lemma cannot be applied because its hypothesis is not satisfied; and indeed the denotation is not closed P-i.j. since it contains the play $q^5 q^3$ but we have $\operatorname{ord}_{(o^1 \rightarrow o^2)\times o^3} q^3 = 0 < 1 = \operatorname{ord}_{o^4 \rightarrow o^5} q^5$.

Observe that the "P-incremental justification" property is preserved across the *curry* isomorphism, but this is not the case for closed P-incremental justification. It is possible to have two isomorphic strategies σ and μ such that one is closed P-i.j. but not the other. For instance any strategy σ that is P-i.j. on the game $I \multimap A$ is also closed P-i.j. When seen as a strategy on the isomorphic game A, however, σ is not necessarily closed P-i.j.[1]; thus the distinction between the games $I \multimap A$ and A matters. This is because the definition of closed P-i.j. strategy specifically refers to the moves of the arena in the left-hand side of the function space arrow $\multimap$. A consequence of this is that the category of closed P-i.j. strategies, that we will introduce later on in this chapter, is neither monoidal closed nor cartesian closed.

6.2.5 Interaction sequences

In this section we recall some basic definitions and results used in game semantics and we fix some notations that will be used to analyze interaction sequences.

We consider the composition of two strategies $\sigma : A \multimap B$ and $\mu : B \multimap C$ as defined in Sec. 2.3.2.6. Figure 6.1 shows the structure of an interaction sequence from $\sigma \parallel \mu$. There are four states represented by the rectangular boxes. The content of the state shows who is to play in each of the game $A \multimap B$, $B \multimap C$ and $A \multimap C$. For instance in state OPP, it is O's turn to

[1]In particular, every P-i.j. strategy σ on the game $!A_1 \otimes \ldots \otimes !A_n \multimap B$, is isomorphic, up to arena-tagging of the moves, to the closed P-i.j. strategy $\Lambda^n(\sigma)$ on the game $I \multimap (A_1, \ldots, A_n, B)$, where Λ denotes the *curry* isomorphism.

play in $A \multimap B$ and P's turn to play in $B \multimap C$ and $A \multimap C$. Arrows represent the moves. When specifying interaction sequence, the following bullet symbols are used to represent moves: ○ for P-moves, ● for O-moves, ◑ for a move playing the role of P in $A \multimap B$ and O in $B \multimap C$ and ◐ for the symmetric of ◑. We sometimes add a subscript to the symbols ○ and ● to denote the component in which the moves is played (A or C).

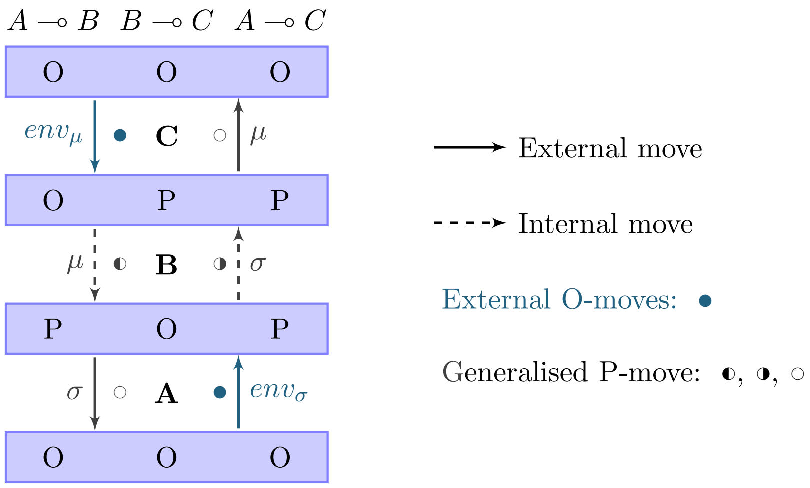


Figure 6.1: Structure of an interaction sequence.

Note that in state OPP, the alternation condition in each of the three games involved prevents the players from playing in A. Indeed, the O-moves in component A of $A \multimap B$ are also O-moves in component A of $A \multimap C$, but the state name indicates that the next move in $A \multimap B$ must be an O-move and the next move in $A \multimap C$ must be a P-move.

Similarly, in the top state OOO, the players cannot make a move in B since the O-moves in component B of the game $B \multimap C$ correspond to P-moves in the component B of $A \multimap B$, but the state name indicates that the next move in $A \multimap B$ and the next move in $B \multimap C$ must be played by O.

Let $u \in Int(A, B, C)$ and m be a move of u. The ***component*** of m is A, B if after playing m the game is under the control of the strategy σ, and B, C otherwise (*i.e.*, if μ has control). In other words, the moves ●, ○ $\in A$ and ◐ $\in B$ shown on the diagram of Fig. 6.1 have component A, B and ●, ○ $\in C$ and ◑ $\in B$ have component B, C.

Also we call ***generalized O-move in component*** A, B moves that play the role of O in the game $A \multimap B$; that is to say moves represented by ◐ and $\bullet_A$. Similarly ◑-moves and $\circ_A$-moves are the ***generalized P-moves in component*** A, B, $\bullet_C$-moves and ◑-moves are the ***generalized O-moves in component*** B, C and $\circ_C$-moves and ◐-moves are the ***generalized P-moves in component*** B, C.

The ***P-view*** of an interaction sequence $u \in Int(A, B, C)$ (also called *core* [McC96b]), written $\overline{u}$ or $\ulcorner u \urcorner$, is defined as:

$$\begin{array}{rlr}
\ulcorner u \cdot n \urcorner &= n & \text{if } n \text{ is an external O-move initial in C,}\\
\ulcorner u \cdot m \cdot v \cdot n \urcorner &= \ulcorner u \cdot m \urcorner \cdot n & \text{if } n \text{ is an external O-move non initial in C,}\\
\ulcorner u \cdot n \urcorner &= \ulcorner u \urcorner \cdot n & \text{if } n \text{ is a generalised P-move.}
\end{array}$$

Lemma 6.33. *Let u be an interaction sequence in $Int(A, B, C)$ then*

$$\ulcorner u \urcorner \upharpoonright A, C = \ulcorner u \upharpoonright A, C \urcorner \ .$$

Proof. By induction on u. It is trivial for the empty sequence. Let b be a move in B. We have $\ulcorner u \cdot b \urcorner \upharpoonright A, C = \ulcorner u \urcorner \upharpoonright A, C$. By the I.H. this equals $\ulcorner u \upharpoonright A, C \urcorner = \ulcorner u \cdot b \upharpoonright A, C \urcorner$. Let

m be a P-move in A or C then $\ulcorner u \cdot m \urcorner \upharpoonright A, C = (\ulcorner u \urcorner \upharpoonright A, C) \cdot m$ and by the I.H. it equals $\ulcorner u \upharpoonright A, C \urcorner \cdot m = \ulcorner (u \upharpoonright A, C) \cdot m \urcorner = \ulcorner u \cdot m \upharpoonright A, C \urcorner$. Let c be an initial move in C. We have $\ulcorner u \cdot c \upharpoonright A, C \urcorner = \ulcorner (u \upharpoonright A, C) \cdot c \urcorner = c = c \upharpoonright A, C = \ulcorner u \cdot c \urcorner \upharpoonright A, C$. Let $u = u_1 \cdot m \cdot u_2 \cdot n$ with n an O-move in $A \to C$. Then necessarily $m \in A, C$ and $\ulcorner u \upharpoonright A, C \urcorner = \ulcorner u_1 \upharpoonright A, C \cdot m \cdot u_2 \upharpoonright A, C \cdot n \urcorner = \ulcorner u_1 \upharpoonright A, C \urcorner \cdot m \cdot n$. Finally by the I.H. this equals $(\ulcorner u_1 \urcorner \upharpoonright A, C) \cdot m \cdot n = (\ulcorner u_1 \urcorner \cdot m \cdot n) \upharpoonright A, C = \ulcorner u_1 \cdot m \cdot u_2 \cdot n \urcorner \upharpoonright A, C$. $\square$

We will also make use of another result that was used by Harmer to show compositionality of P-visible strategies [Har05]:

Lemma 6.34. *[Har05, Lemma 3.3.1] If $u \in Int(A, B, C)$ such that $u \upharpoonright A, B \in \sigma$ and $u \upharpoonright B, C \in \tau$ where σ, τ are two (P-visible) strategies, and m is a generalized O-move of u in component X then $\ulcorner u_{\leqslant m} \upharpoonright X \urcorner = \ulcorner \overline{u_{\leqslant m}} \upharpoonright X \urcorner$.*

Notations 6.35 We now introduce some notations for moves that will come useful when representing plays. The symbol $\bullet$ stands for an O-move and $\circ$ for a P-move. If the game considered is of the form $L \multimap R$ then the we write $\bullet_L$ and $\circ_L$ (resp. $\bullet_R$ and $\circ_R$) to represent a move that belongs to the component L (resp. R). For interaction sequences in $Int(A, B, C)$ we use the set of symbols $\{\bullet_A, \circ_A, \bullet_C, \circ_C, ◐, ◑\}$ as defined in Fig. 6.1. We also identify each of these symbols with the set of moves of the corresponding kind. Thus we write "$m \in \bullet_A$" to mean that m is an O-move played in A. We use the variable X to denote either the component A, B or B, C, and the variable Y to denote the opposite component.

For every given component X, we write $◑_X$ to denote a generalized P-move in X and $◐_X$ to denote a generalized O-move in X. Thus $◐_{A,B} = ◐$, $◑_{A,B} = ◑$, $◐_{B,C} = ◑$, and $◑_{B,C} = ◐$. We write $\bullet_X$ (resp. $\circ_X$) to denote an external O-move (resp. P-move) in component X. Thus $\bullet_{A,B} = \bullet_A$, $\circ_{A,B} = \circ_A$, $\bullet_{B,C} = \bullet_C$, and $\circ_{B,C} = \circ_C$. We write $s \sqsubseteq t$ to say that s is a subsequence (with pointers) of t, $s \leqslant t$ to say that s is a prefix (with pointers) of t and $s \geqslant t$ to say that s is a suffix of t.

6.2.6 Preliminary results

In this section, we prove several preliminary lemmas which will help us study compositionality of P-i.j. strategies.

Lemma 6.36. *Let X be a component (either A, B or B, C). Let u be an interaction sequence of the form $u = \ldots \underset{\circ_X}{\beta} \ldots n \ldots \underset{\bullet_X}{\alpha} \ldots m$ where:*

- *α, β are external moves in component X (necessarily both played in A or in C),*

- *m is either played in B or an external P-move in X,*

- *α is visible at m in X (i.e., $\alpha \in \ulcorner u \upharpoonright X \urcorner$) and consequently β is also visible.*

Then $n \notin \ulcorner u \upharpoonright A, C \urcorner$.

Proof. Since α is an O-move, α and β are necessarily played in the same arena (A or C). Take $v = u$ if m is a generalized O-move in X and $v = u_{<m}$ otherwise (if m is a generalized P-move in X). The third assumption implies $\alpha, \beta \in \ulcorner v \upharpoonright X \urcorner$. The last move in v is necessarily a generalized O-move in component X (see diagram of Fig. 6.1) therefore by Lemma 6.34 we have $\ulcorner v \upharpoonright X \urcorner = \ulcorner \overline{v} \upharpoonright X \urcorner \sqsubseteq \overline{v} \sqsubseteq \overline{u}$. Thus $\alpha, \beta \in \overline{u}$ and since α, β are played in A, C we have $\alpha, \beta \in \overline{u} \upharpoonright A, C = \ulcorner u \upharpoonright A, C \urcorner$ (Lemma 6.33). Finally since n lies underneath the β-α PO-arc it cannot appear in the P-view $\ulcorner u \upharpoonright A, C \urcorner$. $\square$

Lemma 6.37. *Let $u \in Int(A, B, C)$ and n be a move of u such that $n \in \ulcorner u \upharpoonright A, C \urcorner$.*

(i) If all the moves in $u_{\geqslant n}$ are played in C then $n \in \ulcorner u \upharpoonright B, C \urcorner$.

(ii) If all the moves in $u_{\geqslant n}$ are played in A then $n \in \ulcorner u \upharpoonright A, B \urcorner$.

Proof. (i) We show the contrapositive. Suppose that $n \notin \ulcorner u \upharpoonright B, C \urcorner$ then either:
- $\ulcorner u \upharpoonright B, C \urcorner$ contains an initial move $c_0 \in C$ occurring after n in u.
  By Lemma 6.34 we have $\ulcorner u \upharpoonright B, C \urcorner = \ulcorner \overline{u} \upharpoonright B, C \urcorner \sqsubseteq \ulcorner u \urcorner$, thus c_0 also occurs in $\ulcorner u \urcorner$. Since c_0 belongs to C we have $c_0 \in \ulcorner u \urcorner \upharpoonright A, C = \ulcorner u \upharpoonright A, C \urcorner$ (Lemma 6.33). Thus the P-view $\ulcorner u \upharpoonright A, C \urcorner$ starts with the initial move c_0, and since n occurs before c_0 it does not occur in the P-view.
- or n lies underneath a PO-arc β-α visible at $u \upharpoonright B, C$. By assumption, since α occurs after n in u, it must belong to C. We can therefore apply Lemma 6.36 with $X \leftarrow B, C$ which gives $n \notin \ulcorner u \upharpoonright A, C \urcorner$.

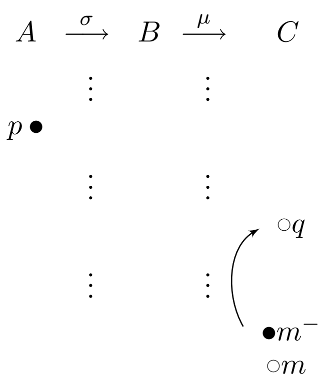
(ii) Suppose that $n \notin \ulcorner u \upharpoonright A, B \urcorner$ then either:
- $\ulcorner u \upharpoonright A, B \urcorner$ contains an initial move $b_0 \in B$ occurring after n in u. But this is impossible since by assumption all the moves occurring after n in u belong to A;
- or n lies underneath a PO-arc β-α in A, B. By assumption, since α occurs after n it must belong to A. We can then conclude using Lemma 6.36 with $X \leftarrow A, B$. $\square$

Note that we cannot completely relax the assumption which says that moves in $u_{\geqslant n}$ are all in the same component. For instance take $u = \bullet_C \; \LEFTcircle \; \underset{n}{\circ_A} \; \LEFTcircle$ then we have $n \in \ulcorner u \upharpoonright A, C \urcorner$ but $n \notin \ulcorner u \upharpoonright A, B \urcorner$.

Lemma 6.38 (P-visibility decomposition (from C))**.** *Let $u = \ldots p \cdot r \cdot m \in Int(A, B, C)$ where p is a $\bullet_A$-move satisfying $p \in \ulcorner u \upharpoonright A, C \urcorner$ and m is in $\circ_C \cup \LEFTcircle \cup \RIGHTcircle$. Then there is a $\RIGHTcircle$-move γ in $r \cdot m$ such that $\gamma \in \ulcorner u \upharpoonright B, C \urcorner$, $p \in \ulcorner u_{\leq \gamma} \upharpoonright A, B \urcorner$ and γ is justified by a move occurring before p.*

Proof. By induction on $|r|$. If $r = \epsilon$ then necessarily $u = \ldots \underset{p}{\bullet_A} \; \underset{m}{\RIGHTcircle}$ where m points before p (since p belongs to A it cannot justify m which is played in B) so we just need to take $\gamma = m$. If $|r| = 1$ then either $u = \ldots \underset{p}{\bullet_A} \; \RIGHTcircle \; \underset{m}{\circ_C}$ or $u = \ldots \underset{p}{\bullet_A} \; \RIGHTcircle \; \underset{m}{\LEFTcircle}$. In both cases we can take γ to be the $\RIGHTcircle$-move between p and m. Suppose $|r| > 1$. Let m^- denote the move preceding m in u. We proceed by case analysis:

- Suppose $m \in \circ_C$ and $m^- \in \bullet_C$. Let q be the external P-move that justifies m^-. Since $p \in \ulcorner u \upharpoonright A, C \urcorner$, q must occur after p in u:



  Thus we can use the induction hypothesis with $u \leftarrow u_{\leqslant q}$: There is a $\RIGHTcircle$-move γ in $u_{]p,q]}$ pointing before p such that $\gamma \in \ulcorner u_{\leqslant q} \upharpoonright B, C \urcorner$, $p \in \ulcorner u_{\leqslant \gamma} \upharpoonright A, B \urcorner$. Moreover $\ulcorner u_{\leqslant q} \upharpoonright B, C \urcorner \leqslant \ulcorner u_{\leqslant m} \upharpoonright B, C \urcorner$ (since q is visible from m in B, C) thus we have $\gamma \in \ulcorner u_{\leqslant m} \upharpoonright B, C \urcorner$ as required.

- Suppose $m \in \circ_C$ and $m^- \in ◑$. Again we can conclude using the I.H. with $u \leftarrow u_{\leqslant m^-}$.

- Suppose $m \in ◑$. There are two cases:

  Either all the moves in r are in A and then r is of the form $(\circ_A \bullet_A)^*$ (where $(\cdot)^*$ denotes the Kleenee star operator). We just need to take $\gamma = m$. Indeed, moves in $u_{\geqslant m}$ are all in A and by assumption $p \in \ulcorner u \upharpoonright A, C \urcorner$ therefore Lemma 6.37(ii) gives $p \in \ulcorner u \upharpoonright A, B \urcorner$. Also, since m is a ◑-move, its justifier is a ◐-move but r contains only $\bullet$ and $\circ$ moves hence m's justifier must occur before p.

  Or r contains at least one move in B. Let b be the last such move, then u is of the form $\ldots p \cdot \ldots \cdot \underset{b}{◐} \cdot (\circ_A \bullet_A)^* \cdot \underset{m}{◑}$. We then have $u \upharpoonright B, C = \ldots p \cdot \ldots \cdot \underset{b}{◐} \cdot \underset{m}{◑}$ thus $b \in \ulcorner u \upharpoonright B, C \urcorner$. We can then conclude by applying the I.H. with $u \leftarrow u_{\leqslant b}$.

- Suppose $m \in ◑$. If $m^- \in ◐$ then the I.H. with $u \leftarrow u_{\leqslant m^-}$ permits us to conclude. If $m^- \in \bullet_C$ then we conclude by applying the I.H. on $u \leftarrow u_{\leqslant q}$ where q is the external P-move in C justifying m^-. $\square$

We now show the symmetric of the previous lemma:

Lemma 6.39 (P-visibility decomposition (from A))**.** *Let* $u = \ldots p \cdot r \cdot m \in Int(A, B, C)$ *where* p *is an O-move* non initial *in* C *satisfying* $p \in \ulcorner u \upharpoonright A, C \urcorner$ *and* m *is in* $\circ_A \cup ◐ \cup ◑$*. Then there is a* ◐*-move* γ *in* $r \cdot m$ *such that* $\gamma \in \ulcorner u \upharpoonright A, B \urcorner$ *,* $p \in \ulcorner u_{\leq \gamma} \upharpoonright B, C \urcorner$ *and* γ *is justified by a move occurring before* p*.*

Proof. The proof is almost symmetrical to the previous one. We proceed by induction on $|r|$. If $r = \epsilon$ then necessarily $u = \ldots \underset{p}{\bullet_C} \underset{m}{◐}$ where m points before p (it cannot point to p since p is not initial in C). Thus we just need to take $\gamma = m$.

If $|r| = 1$ then either $u = \ldots \underset{p}{\bullet_C} ◐ \underset{m}{\circ_A}$ or $u = \ldots \underset{p}{\bullet_C} ◐ \underset{m}{◑}$. In both cases we can take γ to be the ◐-move between p and m. Suppose $|r| > 1$. Let m^- denote the move preceding m in u. We do a case analysis:

- Suppose $m \in \circ_A$ and $m^- \in \bullet_A$. Let q be the external P-move that justifies m^-. Since $p \in \ulcorner u \upharpoonright A, C \urcorner$, q must occur after p in u:

$$
\begin{array}{ccccc}
A & \xrightarrow{\sigma} & B & \xrightarrow{\mu} & C \\
 & \vdots & & \vdots & \\
 & & & & \bullet\ p \\
 & \vdots & & \vdots & \\
q\ \circ & & & & \\
 & \vdots & & \vdots & \\
m^-\ \bullet & & & & \\
m\ \circ & & & &
\end{array}
$$

  Thus we can use the induction hypothesis with $u \leftarrow u_{\leqslant q}$: There is a ◐-move γ in $u_{]p,q]}$ pointing before p such that $\gamma \in \ulcorner u_{\leqslant q} \upharpoonright A, B \urcorner$, $p \in \ulcorner u_{\leqslant \gamma} \upharpoonright B, C \urcorner$. Moreover $\ulcorner u_{\leqslant q} \upharpoonright A, B \urcorner \leqslant \ulcorner u_{\leqslant m} \upharpoonright A, B \urcorner$ (since q is visible from m in A, B) thus we have $\gamma \in \ulcorner u_{\leqslant m} \upharpoonright A, B \urcorner$ as required.

- Suppose $m \in \circ_A$ and $m^- \in ◑$ then again we can conclude using the I.H. with $u \leftarrow u_{\leqslant m^-}$.

- Suppose $m \in ◐$. Then there are two cases: If r does not contain any move in B then it is of the form $(\circ_C \bullet_C)^*$. We just need to take $\gamma = m$. Indeed:

(i) By Lemma 6.37(i) we have $p \in \ulcorner u \upharpoonright B, C \urcorner$.

(ii) the justifier of m occurs before p. Indeed, if m is justified by a $\RIGHTcircle$-move then since $p \cdot r$ contains only $\bullet$ and $\circ$-moves, m's justifier must occur before p; otherwise m's justifier is an initial $\bullet_C$-move c_i which by P-visibility occurs in $\ulcorner u \upharpoonright B, C \urcorner$, but since the P-view computation "stops" when reaching an initial moves, and because p also belongs to the P-view (by (i)), n necessarily occurs after c_i.

Otherwise r contains some move in B. Let b be the last such move. Then u is of the form $u = \ldots p \cdot \ldots \cdot \underset{b}{\LEFTcircle} \cdot (\circ_A \bullet_A)^* \cdot \underset{m}{\RIGHTcircle}$. So we have $u \upharpoonright B, C = \ldots p \cdot \ldots \cdot \underset{b}{\LEFTcircle} \cdot \underset{m}{\RIGHTcircle}$ hence $b \in \ulcorner u \upharpoonright B, C \urcorner$. We can now conclude by applying the I.H. with $u \leftarrow u_{\leqslant b}$.

- Suppose $m \in \RIGHTcircle$. If $m^- \in \RIGHTcircle$ then the I.H. with $u \leftarrow u_{\leqslant m^-}$ permits us to conclude. If $m^- \in \bullet_A$ then we conclude by applying the I.H. on $u \leftarrow u_{\leqslant q}$ where q is the external P-move in A justifying m^-. $\square$

Using the two preceding lemmas we can show the following key result:

Lemma 6.40 (Increasing order lemma)**.** *Let $u = \ldots p \cdot r \cdot m \in Int(A, B, C)$ where*

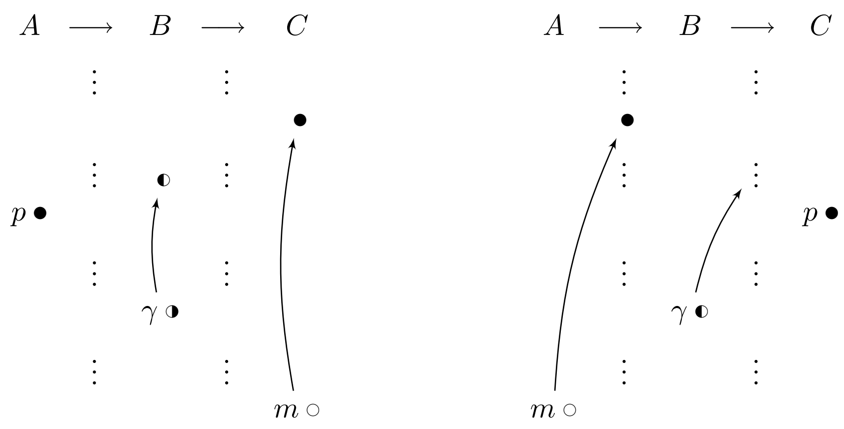
1. *p is an external O-move in component X ($p \in \bullet_A$ and $X = A, B$, or $p \in \bullet_C$ and $X = B, C$) non initial in C,*
2. *$p \in \ulcorner u \upharpoonright A, C \urcorner$,*
3. *m is either played in B (in $\LEFTcircle$ or $\RIGHTcircle$) or is an external P-move in Y (in $\circ_C$ if $p \in \bullet_A$, or in $\circ_A$ if $p \in \bullet_C$),*
4. *m's justifier occurs before p,*
5. *$u \upharpoonright Y$ is P-i.j.,*
6. *$u_{\leqslant b} \upharpoonright X$ is P-i.j. for every B-move b occurring in u such that b is a generalized P-move in X and is not initial in B.*

Then:

$$\underset{Y}{\operatorname{ord}} m \geq \underset{A \multimap C}{\operatorname{ord}} p \ .$$

Proof. If $p \in \bullet_C$ (resp. if $p \in \bullet_A$) then by the second hypothesis we can apply Lemma 6.39 (resp. Lemma 6.38): there is an occurrence in $r \cdot m$ of a non-initial B-move γ of type $\LEFTcircle$ (resp. $\RIGHTcircle$) such that $\gamma \in \ulcorner u \upharpoonright Y \urcorner$, $p \in \ulcorner u_{\leq \gamma} \upharpoonright X \urcorner$ and γ is justified by a move occurring before p.

By hypotheses 1 and 3, there are six possible cases depending on the type of the moves p and m: $(p, m) \in \bullet_A \times (\circ_C \cup \LEFTcircle \cup \RIGHTcircle) \cup \bullet_C \times (\circ_A \cup \LEFTcircle \cup \RIGHTcircle)$. The following diagram illustrates the cases $(p, m) \in \bullet_A \times \circ_C$ (left) and $(p, m) \in \bullet_C \times \circ_A$ (right):



We have:

$$\operatorname*{ord}_{Y} \gamma \geq \operatorname*{ord}_{X} \gamma \ . \tag{6.2}$$

Indeed if $p \in \bullet_C$ then $X = B, C$ and $Y = A, B$ and by Lemma 6.22 we have $\mathrm{ord}_{A \multimap B}\, \gamma \geq \mathrm{ord}_{B \multimap C}\, \gamma$. If $n \in \bullet_A$ then γ is a $◑$-move therefore it is not initial in B and Lemma 6.22 gives $\mathrm{ord}_{A \multimap B}\, \gamma = \mathrm{ord}_{B \multimap C}\, \gamma$.

Hence:

$$\begin{aligned}
\operatorname*{ord}_{A \multimap C} p = \operatorname*{ord}_{X} p & & \text{by Lemma 6.22 since } p \text{ is non initial in } C \\
\leq \operatorname*{ord}_{X} \gamma & & \text{since } u_{\leqslant \gamma} \upharpoonright X \text{ is P-i.j. (hyp. 6) and } \gamma\text{'s justifier occurs before } p \\
\leq \operatorname*{ord}_{Y} \gamma & & \text{by (6.2)} \\
\leq \operatorname*{ord}_{Y} m & & \text{since } u \upharpoonright Y \text{ is P-i.j. (hyp. 5) and } m\text{'s justifier occurs before } \gamma \text{ (hyp. 4).} \ \square
\end{aligned}$$

Remark 6.41 The above lemma can be viewed as a semantic translation of the abstraction typing rule of the safe lambda calculus. Hypotheses 5 and 6 correspond to the P-incremental justification requirement. Hypothesis 3 says that m is a generalized P-move. By the Correspondence Theorem (see Chapter 5) such moves correspond to variable nodes in the computation tree of the term considered. Similarly, hypothesis 1 says that p corresponds to a binder node. Finally, using McCusker's terminology [MP07], hypotheses 2 and 4 say that m is an *external move to* p. This expresses semantically the notion of free variable: the variable corresponding to m occurs free in the subterm rooted at the binder node corresponding to p. In other words, the lemma states that after abstracting variables, the order of a free variable is greater than the order of the term formed. This is precisely the definition of the abstraction rule of the safe lambda calculus.

Lemma 6.42. *Let $u \in Int(A, B, C)$ such that $u = \ldots \gamma \ldots \delta \ldots m$ where m is a generalized P-move in X, $\gamma \in \ulcorner u \upharpoonright A, C \urcorner$ and $\delta \in \ulcorner u \upharpoonright X \urcorner$. Then $\gamma \in \ulcorner u_{\leqslant \delta} \upharpoonright A, C \urcorner$.*

Proof. First we remark that δ must occur in $\ulcorner u \urcorner$. Indeed, $\delta \in \ulcorner u \upharpoonright X \urcorner = \ulcorner u_{<m} \upharpoonright X \urcorner \cdot m$ therefore $\delta \in \ulcorner u_{<m} \upharpoonright X \urcorner$ and since the move preceding m in u is necessarily a generalized O-move in X, we can apply Lemma 6.34:

$$\begin{aligned}
\delta \in \ulcorner u_{<m} \upharpoonright X \urcorner &= \ulcorner \ulcorner u_{<m} \urcorner \upharpoonright X \urcorner & & \text{by Lemma 6.34} \\
&\sqsubseteq \ulcorner u_{<m} \urcorner \\
&\sqsubseteq \ulcorner u \urcorner \ .
\end{aligned}$$

Clearly, $\ulcorner u_{\leqslant \delta} \upharpoonright A, C \urcorner$ is a prefix of $\ulcorner u \upharpoonright A, C \urcorner$, indeed:

$$\begin{aligned}
\ulcorner u_{\leqslant \delta} \upharpoonright A, C \urcorner &= \ulcorner u_{\leqslant \delta} \urcorner \upharpoonright A, C & & \text{by Lemma 6.33} \\
&\leqslant \ulcorner u \urcorner \upharpoonright A, C & & \text{since } \delta \in \ulcorner u \urcorner \\
&= \ulcorner u \upharpoonright A, C \urcorner & & \text{by Lemma 6.33.}
\end{aligned}$$

Finally since $\gamma \in \ulcorner u \upharpoonright A, C \urcorner$ and γ occurs before δ in u, we necessarily have $\gamma \in \ulcorner u_{\leqslant \delta} \upharpoonright A, C \urcorner$. $\square$

Lemma 6.43. *Let X be a component and $u \in Int(A, B, C)$ such that the projection of u on the component X has the form:*

$$u \upharpoonright X = \ldots n \ldots \underset{\bullet_X}{p} \ \ldots \underset{\circ_X}{m}$$

and

1. *m and p are external move in X (in A if $X = A, B$ and in C if $X = B, C$),*

2. $u \upharpoonright X$ *is P-i.j.,*

3. $u_{\leqslant b} \upharpoonright A, B$ *is P-i.j. for every* ◑*-move* b *occurring in* u*,*

4. $u_{\leqslant b} \upharpoonright B, C$ *is P-i.j. for every* ◐*-move* b *not initial in* B *occurring in* u*.*

Then either $\operatorname{ord}_{A\multimap C} p \leq \operatorname{ord}_{A\multimap C} m$ *or* $p \notin \ulcorner u \upharpoonright A, C \urcorner$*.*

Proof. - Suppose that p occurs in the P-view $\ulcorner u \upharpoonright X \urcorner$. Then we have

$$\operatorname*{ord}_{A\multimap C} p = \operatorname*{ord}_{B\multimap C} p \ . \tag{6.3}$$

Indeed, if X is the component B, C then necessarily p is not initial in C (otherwise it would be the first move in $\ulcorner u \upharpoonright B, C \urcorner$, which is not the case since by visibility n occurs before p in the P-view) and if $X = A, B$ then p is in A. In both cases, Lemma 6.22 gives us the claimed equality.
Thus,

$$\begin{aligned}
\operatorname*{ord}_{A\multimap C} p &= \operatorname*{ord}_{X} p && \text{by (6.3)} \\
&\leq \operatorname*{ord}_{X} m && \text{since } u \upharpoonright X \text{ is P-i.j.} \\
&= \operatorname*{ord}_{A\multimap C} m && \text{by Lemma 6.22 since } m \text{ is not initial in } C.
\end{aligned}$$

- Suppose that p does not occur in the P-view $\ulcorner u \upharpoonright X \urcorner$, then p lies underneath a PO arc occurring in $\ulcorner u \upharpoonright X \urcorner$. We denote this arc by β-α where β and α denote the arc's nodes. We have:

$$u \upharpoonright X = \ldots n \ldots \beta \ldots p \ldots \alpha \ldots m$$

with $\operatorname{ord}_X \alpha \leq \operatorname{ord}_X m$ (since $u \upharpoonright X$ is P-i.j.).
– Suppose α is an external move then so is β. Indeed, if $X = B, C$ and $\alpha \in \bullet_C$ then α can only point to another move in C and if $X = A, B$ and $\alpha \in \bullet_A$ then since α is an O-move in A, B, it is not initial in A and therefore its justifier must also be in A. Instancing Lemma 6.36 with $n \leftarrow p$ gives us $p \notin \ulcorner u \upharpoonright A, C \urcorner$.
– Suppose α is a B-move then necessarily so is β (Indeed, if $X = A, B$ then $\alpha \in B$ can only point to a move in B; if $X = B, C$ then since α is an O-move in the game B, C it is not initial in B so its justifier must also be in B). Suppose that $p \in \ulcorner u \upharpoonright A, C \urcorner$, then applying Lemma 6.42 with $\delta, \gamma \leftarrow \alpha, p$ gives $p \in \ulcorner u_{\leqslant \alpha} \upharpoonright A, C \urcorner$. By the 3^{rd} and 4^{th} hypothesis, $u_{\leqslant \alpha} \upharpoonright X$ is P-i.j. and we can use Lemma 6.40 on $u_{\leqslant \alpha}$:

$$\begin{aligned}
\operatorname*{ord}_{A\multimap C} p &\leq \operatorname*{ord}_{Y} \alpha && \text{by Lemma 6.40 with } u \leftarrow u_{\leqslant \alpha} \\
&= \operatorname*{ord}_{X} \alpha && \text{by Lemma 6.22 since } \alpha \text{ is a non initial } B\text{-move} \\
&\leq \operatorname*{ord}_{X} m && \text{since } u \upharpoonright X \text{ is P-i.j.} \\
&= \operatorname*{ord}_{A\multimap C} m && \text{by Lemma 6.22 since } m \text{ is not initial in } C. \ \square
\end{aligned}$$

Linear composition

Proposition 6.44 (Linear composition). *Let* $\sigma : A \multimap B$ *and* $\mu : B \multimap C$ *be two well-bracketed (P-visible) strategies then*

(i) σ *closed P-i.j.* $\wedge$ μ *P-i.j.* $\implies$ $\sigma;\mu$ *P-i.j.;*

(ii) σ *and* μ *are closed P-i.j.* $\implies$ $\sigma;\mu$ *closed P-i.j.*

Proof. Since well-bracketing is preserved by strategy composition [AMJ94, Proposition 2.5], $\sigma;\mu$ is well-bracketed so we can use the definition of P-i.j. from Proposition 6.24.
(i) We prove that $\sigma;\mu$ is P-i.j. Let u be a play of the interaction $\sigma \parallel \mu$ ending with an external P-move m justified by n in $\ulcorner u \upharpoonright A, C \urcorner$. Let p be an external O-move occurring between n and m:

$$u \upharpoonright A, C = \ldots \underset{\bullet}{n} \ldots \underset{\bullet}{p} \ldots \underset{\circ}{m} \ .$$

To show that $u \upharpoonright A, C$ is P-incrementally justified, we just need to prove that either $p \notin \ulcorner u \upharpoonright A, C \urcorner$ or $\mathrm{ord}_{A \multimap C}\, p \leq \mathrm{ord}_{A \multimap C}\, m$. Note that if $p \in \ulcorner u \upharpoonright A, C \urcorner$ then necessarily p is not initial in C because n occurs before p in $\ulcorner u \upharpoonright A, C \urcorner$.

Let E denote one of the two external arenas (A or C), X be the corresponding component (*i.e.*, $X = A, B$ if $E = A$, and $X = B, C$ if $E = C$) and Y denote the other component.

1) Suppose m and n are two external moves in E.

   1.a) Suppose $p \in E$. This situation is handled by Lemma 6.43: we have either $\mathrm{ord}_{A \multimap C}\, p \leq \mathrm{ord}_{A \multimap C}\, m$ or $p \notin \ulcorner u \upharpoonright A, C \urcorner$.

   1.b) Suppose $p \notin E$. If $p \in \ulcorner u \upharpoonright A, C \urcorner$, then by Lemma 6.40 with $X \leftarrow Y$ we have $\mathrm{ord}_{A \multimap C}\, p \leq \mathrm{ord}_X\, m$ and since m is not initial in C Lemma 6.22 gives $\mathrm{ord}_X\, m = \mathrm{ord}_{A \multimap C}\, m$ thus $\mathrm{ord}_{A \multimap C}\, p \leq \mathrm{ord}_{A \multimap C}\, m$.

2) Suppose $m \in A$ and $n \in C$. Then m is an initial move in A pointing to a $\mathbin{\bullet\!\!\circ}$-move b_0 initial in B which in turn points to the $\bullet_C$-move n initial in C.

   This situation differs from the previous case because the justifier of m in the game A, C differs from its justifier in A, B (see Sec. 2.3.2.6 for the definition of projection on the overall component A, C), thus it is not guaranteed that m's justifier in A, C occurs before p so we cannot use Lemma 6.40.

   Let's assume that $p \in \ulcorner u \upharpoonright A, C \urcorner$ and prove that $\mathrm{ord}_{A \multimap C}\, p \leq \mathrm{ord}_{A \multimap C}\, m$.

   - Suppose p occurs before b_0 (in which case we cannot use Lemma 6.40). Up to now we have only used the fact that σ and μ are P-i.j. The assumption that σ is *closed* P-i.j. now becomes crucial.

     Since $p \in \ulcorner u \upharpoonright A, C \urcorner$ and $b_0 \in \ulcorner u \upharpoonright B, C \urcorner$, applying Lemma 6.42 with $X \leftarrow B, C$ and $\delta, \gamma \leftarrow b_0, p$ gives $p \in \ulcorner u_{\leqslant b_0} \upharpoonright A, C \urcorner$. This allows us to apply Lemma 6.40 on $u_{\leqslant b_0}$:

$$\begin{aligned}
\underset{A \multimap C}{\mathrm{ord}}\, m = \underset{A}{\mathrm{ord}}\, m \geq \underset{B}{\mathrm{ord}}\, b_0 & \qquad \text{since } u \upharpoonright A, B \text{ is closed P-i.j. and } m \text{ is initial in } A \\
= \underset{B \multimap C}{\mathrm{ord}}\, b_0 & \\
\geq \underset{A \multimap C}{\mathrm{ord}}\, p & \qquad \text{by Lemma 6.40 on } u_{\leqslant b_0} \text{ with } X \leftarrow A, B.
\end{aligned}$$

   - Suppose p occurs after b_0 (and necessarily before m).

     a. Suppose $p \in C$. Then m's justifier occurs before p in u thus by Lemma 6.40 we have $\mathrm{ord}_{A \multimap C}\, p \leq \mathrm{ord}_{A \multimap B}\, m = \mathrm{ord}_{A \multimap C}\, m$.

     b. Suppose $p \in A$. Since $p \in \ulcorner u \upharpoonright A, C \urcorner$, by Lemma 6.43 with $X \leftarrow A, B$ and $(n, p, m) \leftarrow (b_0, p, m)$ we have $\mathrm{ord}_{A \multimap C}\, p \leq \mathrm{ord}_{A \multimap C}\, m$.

     (Note that: (i) we cannot use Lemma 6.40 on u because m and p are both played in A; (ii) in the case where A has a single initial move then p is necessarily hereditarily enabled by the initial move m and we can immediately conclude that $\mathrm{ord}_{A \multimap C}\, p \leq \mathrm{ord}_{A \multimap C}\, m$. This argument does not work in the general case however.)

(ii) We now show that $\sigma;\mu$ is closed P-i.j. provided that both σ and μ are. Take a play $sm \in \sigma;\mu$ such that m is initial in A and let n be the initial move of C justifying m. Let $u \in \sigma \parallel \mu$ be the uncovering of sm ($sm = u \upharpoonright A, C$) and b_0 be the initial B-move justifying m in u. We have:

$$\begin{aligned} \underset{A}{\mathrm{ord}}\, m &\geq \underset{B}{\mathrm{ord}}\, b_0 && \text{since } u \upharpoonright A, B \in \sigma \text{ is closed P-i.j.} \\ &\geq \underset{C}{\mathrm{ord}}\, n && \text{since } u_{\leqslant b_0} \upharpoonright B, C \in \mu \text{ is closed P-i.j..} \ \square \end{aligned}$$

Observe that the second part of the proposition gives only a *sufficient* condition for $\sigma;\mu$ to be closed P-i.j.: we can have $\sigma;\mu$ closed P-i.j. although μ is not.

Tensor product

Given two strategies $\sigma : A \multimap B$ and $\tau : C \multimap D$, their tensor product is denoted $\sigma\otimes\tau : A\otimes B \multimap C \otimes D$ where $A \otimes B$ denotes the tensor product of the games A and B (see Sec. 2.3.3.1).

Proposition 6.45. *If $\sigma : A \multimap B$ and $\tau : C \multimap D$ are P-i.j. (resp closed P-i.j.) then so is $\sigma \otimes \tau$.*

Proof. By establishing the state diagram of the game $A \otimes C \multimap B \otimes D$ one can show easily that only player O can switch between the subgames $A \multimap B$ and $C \multimap D$. Consequently, in the P-view of a play of the game $A \otimes C \multimap B \otimes D$, all the moves are played in the same subgame (*i.e.*, all in $A \multimap B$ or all in $C \multimap D$). Hence if the last move of a play m is played in $A \multimap B$ then $\ulcorner s \upharpoonright A, B\urcorner = \ulcorner s \urcorner \upharpoonright A, B = \ulcorner s \urcorner$ (and conversely if m is played in $C \multimap D$). The result follows immediately. $\square$

Pairing and projection

Given two strategies $\sigma : C \multimap A$ and $\tau : C \multimap B$, let $\langle \sigma, \tau\rangle : C \multimap A \times B$ denote the pairing strategy as defined in Sec. 2.3.3.3 where $A \times B$ denotes the product of the games A and B.

Proposition 6.46 (Pairing)**.**

(i) If $\sigma : C \multimap A$ and $\tau : C \multimap B$ are P-i.j. (resp. closed P-i.j.) then so is $\langle\sigma, \tau\rangle$;

(ii) For every objects A and B, the projections $\pi_1 : A \times B \multimap A$ and $\pi_2 : A \times B \multimap B$ are closed P-i.j.

The proof is immediate.

Promotion

Let s be a play. We call ***thread*** a maximal subsequence of s consisting of moves that are hereditarily justified by the same occurrence of an initial move. For every move m occurring in s there is only one thread in s containing it; this thread is called the ***thread of*** m.

Recall that the promotion $\sigma^\dagger : !A \multimap !B$ of a strategy $\sigma : !A \multimap B$, for two well-opened games A and B, is given by:

$$\sigma^\dagger = \{ s \in L_{!A \multimap !B} \mid \text{for all inital } m \text{ in } B,\ s \upharpoonright m \in \sigma \} \ .$$

Since B is well-opened, plays of σ consist of a single thread initiated by some initial B-move. Plays of $\sigma^\dagger$, however, are interleaves of potentially infinitely many single-threaded plays of σ. The visibility condition implies that the thread of a P-move is always the same as the thread of the preceding O-move. Consequently, the P-view of a play is equal to the P-view of the current thread: for any play s we have $\ulcorner s \urcorner = \ulcorner s \upharpoonright b \urcorner = \ulcorner s \urcorner \upharpoonright b$ where $b \in B$ is the initial move opening the current thread of s.

The state of the game is given by an infinite sequence of symbols in $\{O, P\}$, each element of the sequence indicating who is to play in the corresponding thread. The diagram on Fig. 6.2 illustrates how the state changes as a play of $\sigma^\dagger$ unfolds. The initial state of the game is O^ω—an infinite sequence of O's—indicating that O is to play in all the threads. When O plays an initial move in B, it "opens" a new thread so the state of the game becomes $O^k P O^\omega$ where k is the index of the thread being opened. By alternation, P now has to play; his move must be played in a thread already opened by O and in which P is to play. Only one thread is in such state: the k^{th} one; thus when P makes his move the game is set back to state O^ω.

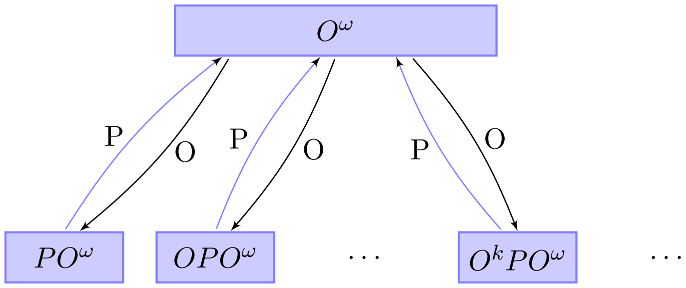


Figure 6.2: State diagram for plays of $\sigma^\dagger$.

Proposition 6.47 (Promotion)**.** *If A and B are two well-opened games and $\sigma : !A \multimap B$ is a well-bracketed P-i.j. strategy then $\sigma^\dagger$ is also well-bracketed and P-i.j. Furthermore if σ is closed P-i.j. then so is $\sigma^\dagger$.*

Proof. $\sigma^\dagger$ is well-bracketed [AMJ94, Proposition 2.10.]. For P-incremental justification, the result is a direct consequence of the fact that the P-view of a play in $\sigma^\dagger$ is equal to the P-view of the current thread. For closed P-incremental justification, the result is immediate. ☐

Composition

We recall that the composite of $\sigma : !A \multimap B$, and $\mu : !B \multimap C$ in the co-Kleisli category of games $\mathcal{C}$ (Def. 2.59), written $\sigma \fatsemi \mu$, is defined as:

$$\sigma \fatsemi \mu = \sigma^\dagger ; \mu \ .$$

From propositions 6.44 and 6.47 we obtain:

Proposition 6.48 (Compositionality)**.** *Let A and B be two well-opened games. Let $\sigma : !A \multimap B$ and $\mu : !B \multimap C$ be two well-bracketed strategies then:*

(i) If σ is closed P-i.j. and μ is P-i.j. then $\sigma \fatsemi \mu : !A \multimap C$ is also P-i.j.;

(ii) If σ and μ are closed P-i.j. then so is $\sigma \fatsemi \mu : !A \multimap C$.

6.2.7 Categories of closed P-i.j. strategies

We define the category of closed P-incrementally justified strategies as follows:

- Objects: games (as defined in Sec. 2.3.2.2),
- Morphisms $A \to B$: closed P-i.j. strategies for the game $A \multimap B$,
- Composition: the linear strategy composition (Def. 2.55).

The results of the previous section show that this is indeed a monoidal category. It is not monoidal closed, however. Indeed, recall that a P-i.j. strategy $\sigma : A \multimap B$ is *closed* P-i.j. if some condition on the initial A-moves occurring in the plays is met. In particular if A has no initial move, σ is necessarily closed P-i.j. Consequently the isomorphic strategy on the game

$I \multimap (A \multimap B)$ obtained by *currying* is closed P-i.j. although σ itself is not necessarily closed P-i.j. Take for instance the two simply-typed terms $\vdash_{\mathsf{st}} \lambda x^o y^o.y$ and $y : o \vdash_{\mathsf{st}} \lambda x^o.y$. These two terms have isomorphic denotations in $\mathcal{C}$. But the denotation of the first term is closed P-i.j. while the second is only P-i.j.

We define the ***intensional category*** $\mathcal{I}$ as the co-Kleisli category of the category defined above.

Intensional category

Let $\mathcal{C}$ denote the co-Kleisli category of games defined in Sec. 2.3.3.6.

Lemma 6.49. *Let* ord *be the order function from Def. 2.74: for every game A with underlying set of moves M_A:*

$$\operatorname{ord} A \stackrel{def}{=} \max_{m \in M_A} \operatorname{ord} m$$

with the convention $\max \emptyset = -1$*. We define the function* dro *on objects of $\mathcal{C}$ as follows. For every game A with underlying set of* initial *moves I_A:*

$$\operatorname{dro} A \stackrel{def}{=} \min_{m \in I_A} \operatorname{ord} m \ .$$

Then the triple $(\mathcal{C}, \operatorname{ord}, \operatorname{dro})$ defines a pre-incremental closed category.

Proof. The functions ord and dro trivially satisfy the conditions of Def. 6.6. $\square$

Proposition 6.50. $(\mathcal{C}, \mathcal{I}, \operatorname{ord}, \operatorname{dro})$ *is an ICC.*

Proof. The objects of $\mathcal{I}$ are exactly those of $\mathcal{C}$. The morphisms of $\mathcal{I}$ are a subclass of morphisms of $\mathcal{C}$. For every object A, the identity strategy id_A is closed P-i.j. For every pair of morphisms in $\mathcal{I}$ the composite is also in $\mathcal{I}$ by Prop. 6.48. Thus $\mathcal{I}$ is a lluf subcategory of $\mathcal{C}$. The empty strategies $\epsilon_A : A \to I$ all belong to $\mathcal{I}$ therefore the empty game $\mathbf{1}$ is terminal in $\mathcal{I}$. By Prop. 6.46, projections are closed P-i.j., and closed P-i.j. strategies are closed under pairing. Because of Lemma 6.31(i) closed P-i.j. strategies are closed under incremental currying. Finally closed P-i.j. strategies are closed under right composition with evaluation maps followed by incremental currying: let $f : A_1 \times A_2 \to C^B$ be a closed P-i.j., since the evaluation map $ev_{B,C}$ is P-i.j. by Proposition 6.44(i) $f; ev_{B_C} : A_1 \times A_2 \to C$ is P-i.j., thus if $\operatorname{dro} A_1 \geq \operatorname{ord}(C^{A_2})$, by Lemma 6.31(i) $\Lambda(f; ev_{B_C}) : A_1 \to C^{A_2}$ is *closed* P-i.j. Hence $(\mathcal{C}, \mathcal{I}, \operatorname{ord}, \operatorname{dro})$ is an ICC. $\square$

The category $\mathcal{I}$ will be used to give the intensional game model of safe PCF and safe IA. We write $\mathcal{I}_{ib}$, $\mathcal{I}_b$ and $\mathcal{I}_i$ to denote its lluf subcategories of innocent, well-bracketed and innocent and well-bracketed strategies respectively.

Extensional category

Let $\lesssim$ denote the usual intrinsic preorder of the category $\mathcal{C}$ (see Sec. 2.3.3.6). The preorder $\lesssim_{\mathcal{I}}$ on morphisms of the category $\mathcal{C}$ is defined similarly to $\lesssim$ except that the test strategy α ranges over the morphisms of the subcategory $\mathcal{I}$ only: for $\sigma, \mu \in \mathcal{C}(I, A)$,

$$\sigma \lesssim_{\mathcal{I}} \tau \quad \iff \quad \forall \alpha \in \mathcal{I}(A, \Sigma).\ \sigma \fatsemi \tau = \top \implies \tau \fatsemi \alpha = \top \ .$$

The ***intrinsic preorder*** in $\mathcal{I}$, also written $\lesssim_{\mathcal{I}}$, is defined as the restriction of $\lesssim_{\mathcal{I}}$ to the morphisms of the category $\mathcal{I}$. Abramsky et al. [AMJ94] proved that $\lesssim$ is a CCC precongruence for $\mathcal{C}$. The same proof shows that $\lesssim_{\mathcal{I}}$ is also a CCC precongruence for $\mathcal{C}$. Consequently by Lemma 6.16, the ***extensional category*** $\mathcal{I}/\lesssim_{\mathcal{I}}$ is a rational ICC.

Interpretation

By Prop. 6.13, we have that the ICCs $\mathcal{I}$ and $\mathcal{I}/\lesssim_{\mathcal{I}}$ both provide a model of the safe lambda calculus, and the rational ICCs $\mathcal{I}_{ib}$ and $\mathcal{I}_{ib}/\lesssim_{\mathcal{I}_{ib}}$ of innocent well-bracketed closed P-i.j. strategies both provide a model of safe PCF.

6.3 Interpretation in the standard game model

In Chapter 5 we have shown by a syntactic argument, based on the theory of traversals, that safe lambda-terms are denoted in the standard game model by P-i.j. strategies. We now reprove this result by a semantic argument based on the results of the previous section.

6.3.1 Safe lambda calculus with product

Proposition 6.51. *In the standard game model of the simply-typed lambda calculus with product, safe terms are denoted by closed P-i.j. strategies.*

Proof. We show by induction on the formation rules that (i) almost safe terms are denoted by P-i.j. strategies; (ii) safe terms are denoted by *closed* P-i.j. strategies.

- (var) $[\![x : A \vdash_{\mathsf{s}} x : A]\!]$ is the identity strategy id_A which is closed P-i.j.
- (wk) Take $\Gamma \subset \Delta$ and suppose $[\![\Gamma \vdash_{\mathsf{s}} s : A]\!]$ is closed P-i.j. Up to a retagging of the moves, the two strategies $[\![\Delta \vdash_{\mathsf{s}} s : A]\!]$ and $[\![\Gamma \vdash_{\mathsf{s}} s : A]\!]$ are isomorphic. Hence $[\![\Delta \vdash_{\mathsf{s}} s : A]\!]$ is P-i.j. It is also closed P-i.j. since none of the new initial moves introduced by Δ occurs in any play of the strategy.
- $(\times)$, (π_1) and (π_2): The result follows from the I.H. and Proposition 6.46.
- (δ): It follows from the I.H.
- $(\mathsf{app_{as}})$ Suppose that $\Gamma \Vdash_{\mathsf{app}} t_0 t_1 \ldots t_n : B$ with $\Gamma \vdash_{\mathsf{s}} t_0 : (A_1, \ldots, A_n, B)$ and $\Gamma \vdash_{\mathsf{s}} t_i : A_i$ for $i \in \{1..n\}$. By the I.H., for $i \in \{0..n\}$ the strategy $[\![t_i]\!]$ is closed P-i.j. We then have $[\![t_0 t_1 \ldots t_n]\!] = \langle [\![t_0]\!], [\![t_1]\!], \ldots, [\![t_n]\!] \rangle ; ev^n$ where ev^n is the n-parameter evaluation strategy. By Proposition 6.46 the strategy $\langle [\![t_0]\!], [\![t_1]\!], \ldots, [\![t_n]\!] \rangle$ is closed P-i.j. Since the evaluation map ev^n is P-i.j. (but not necessarily closed P-i.j.), by Proposition 6.44(i) $[\![\Gamma \vdash_{\mathsf{s}} t_0 t_1 \ldots t_n : B]\!]$ is P-i.j.
- (app) Terms formed with this rule can also be formed with the rule $(\mathsf{app_{as}})$, therefore by the previous case the denotation of the term formed is P-i.j. By the side-condition of the rule, all the prime sub-types of Γ have order greater than the order of the term, therefore by Lemma 6.31(ii), $[\![\Gamma \vdash_{\mathsf{s}} t_0 t_1 \ldots t_n : B]\!]$ is *closed* P-i.j.
- (abs): By the I.H., the premise of the rule has a P-i.j. denotation. The denotation of the term in the conclusion of the rule is isomorphic, up to currying, to the denotation of the premise. Therefore it is also P.i.j. And by the side-condition and Lemma 6.31(ii) this implies that it is *closed* P-i.j. $\square$

6.3.2 Safe PCF

Proposition 6.52. *In the standard game model of PCF, safe terms are denoted by closed P-incrementally justified strategies.*

Proof. We first prove the result for PCF_1—the fragment of PCF containing terms of the form $\Omega_A = \mathsf{Y}(\lambda x^A.x)$ but where no other use of Y is allowed [AM98b]. The proof is by structural induction over the structure of the term:

- the strategy $[\![\Omega_A]\!] = \bot$ is clearly closed P-i.j.;
- the functional rules are treated the same way as in the corresponding proof for the safe lambda calculus;

• for the arithmetic rules, we observe that the strategies *succ*, *pred* and *cond* are all closed P-i.j. The fact that pairing and strategy composition preserve closed P-incremental justification permits us to conclude.

We now lift the result to full PCF using the technique of *syntactic approximant* [AM98b]. We have [AM98b, lemma 16]:

$$[\![M]\!] = \bigcup_{n\in\omega} [\![M_n]\!]$$

where M_n is the PCF$_1$ term obtained from M by replacing each subterm of the form $\mathsf{Y}N$ with $\mathsf{Y}^n N_n$, and $\mathsf{Y}^n F$ denotes the n^{th} approximant of $\mathsf{Y}F$. Since the M_ns are PCF$_1$ terms, by the previous result each $[\![M_n]\!]$ is closed P-i.j. and since closed P-incremental justification is clearly a continuous property, $[\![M]\!]$ is also closed P-i.j. □

6.3.3 Safe Idealized Algol

We now extend the game-semantic interpretation to safe IA. The constants of IA are all denoted by closed P-incrementally justified strategies:

Lemma 6.53.

(i) The strategy denotations of the IA constants `skip`*,* `assign`*,* `deref`*,* `mkvar`*,* $\mathtt{seq}_{\mathtt{exp}}$*, and* $\mathtt{seq}_{\mathtt{com}}$ *are all closed P-i.j.*

(ii) The memory-cell strategy cell $: I \multimap\ !\mathtt{var}$ *is closed P-i.j.*

Proof. (i) Inspecting the view functions of these denotations (as defined in Sec. 2.3.5) reveals that they are indeed all closed P-i.j. (ii) Since the game `var` does not contain any P-question, any strategy on the game $I \multimap\ !\mathtt{var}$ is P-i.j. (and therefore also closed P-i.j.). □

Our game-semantic analysis of safe PCF immediately extends to strongly safe IA:

Proposition 6.54. *Strongly safe IA terms are denoted by closed P-i.j. strategies.*

Proof. The proof is an adaptation of the proof for safe PCF. We first show that the result holds for the fragment of *strongly safe IA* in which the only allowed uses of Y are in terms of the form Ω. By induction on the term's structure: For the functional and arithmetic rules, the proof is the same as for safe PCF. For the imperative rules, the result follows from the fact that IA constants are denoted by closed P-i.j. strategies (Lemma 6.53(i)) and because tensor product and composition both preserve closed P-incremental justification. For the block-allocation construct, the result follows from the fact that *cell* is closed P-i.j. (Lemma 6.53(ii)) and that pairing and strategy composition both preserve closed P-incremental justification.

The result is then lifted to the whole of strongly safe IA using the technique of syntactic approximants as in the PCF case. □

We now want to extend this result to safe IA. This turns out to be slightly more difficult than for the strongly-safe fragment. Indeed, in safe IA the safety restriction only constrains variables from the Γ-context (*i.e.*, those that are bound by a λ-abstraction). The fact that Ξ-variables are not constrained is reflected in the semantics. For instance the denotation of the safe *split*-term $\emptyset | x : \mathtt{var} \vdash_{\mathsf{s}} \lambda f^{\mathtt{exp}\to\mathtt{exp}}.\mathtt{deref}\ x$ is not closed P-i.j.

We show, however, that safe split-terms are denoted by strategies in which all the plays are closed P-i.j. except those containing moves from the Ξ-context. Consequently, by "abstracting" Ξ-variables using the constructs `mkvar` or the block-declaration `new`, we eliminate the plays that are not closed P-i.j. Hence since safe IA terms are the *semi-closed* split-terms (*i.e.*, with an empty Ξ-component), this implies that their denotation is closed P-i.j.

Definition 6.55 (P-i.j. modulo $\mathfrak{M}$)**.** Let σ be a strategy on some game A and $\mathfrak{M}$ be a set of moves. We say that σ is ***P-incrementally justified modulo*** $\mathfrak{M}$ iff every even-length play in σ ending with a question that is not in $\mathfrak{M}$ is P-i.j. Similarly we say that σ is *closed* P-i.j. modulo $\mathfrak{M}$ iff every such play is *closed* P-i.j.

A strategy is thus P-i.j. if and only if it is P-i.j. modulo $\emptyset$.

The common operations on strategies preserve the property of being P-incremental justification modulo a set of moves:

Lemma 6.56 (Composition)**.** *Let $\sigma : A \to B$ and $\mu : B \to C$. Let $\mathfrak{M}$ be any set of moves initial in A. If σ is closed P-i.j. modulo $\mathfrak{M}$ and μ is P-i.j. (resp. closed P-i.j.) then $\sigma \fatsemi \mu$ is P-i.j. (resp. closed P-i.j.) modulo $\mathfrak{M}$.*

Proof. We observe that in the proof of compositionality for closed P-i.j. strategies, to show that a play $u \upharpoonright A, C$ of $\sigma; \mu$ is P-i.j. we did not use the fact that *every* play of σ is P-i.j., but only that $u \upharpoonright A, B$ (resp. $u \upharpoonright B, C$) is P-i.j. and all the prefixes of $u \upharpoonright A, B$ and $u \upharpoonright B, C$ *ending with a non-initial B-move* are P-i.j. Thus the same proof can be used to show that a play $u \upharpoonright A, C$ ending with a move not in $\mathfrak{M}$ is P-i.j. ☐

Lemma 6.57 (Tensor product)**.** *Let $\sigma : A \multimap B$ and $\tau : C \multimap D$. Let $\mathfrak{M}_A$ and $\mathfrak{M}_C$ be two sets of moves initial in A and C respectively.*

1. *If σ and τ are P-i.j. modulo $\mathfrak{M}_A$ and modulo $\mathfrak{M}_C$ respectively then $\sigma \otimes \tau$ is P-i.j. modulo $\mathfrak{M}_A \cup \mathfrak{M}_C$;*

2. *If σ and τ are* closed *P-i.j. modulo $\mathfrak{M}_A$ and modulo $\mathfrak{M}_C$ respectively then $\sigma \otimes \tau$ is* closed *P-i.j. modulo $\mathfrak{M}_A \cup \mathfrak{M}_C$.*

Lemma 6.58 (Pairing)**.** *Let $\sigma : C \multimap A$, $\tau : C \multimap B$, and $\mathfrak{M}_C$ be a sets of moves initial in C.*

(i) If σ and τ are P-i.j. modulo $\mathfrak{M}_C$ then so is $\langle \sigma, \tau \rangle$;

(ii) If σ and τ are closed P-i.j. modulo $\mathfrak{M}_C$ then so is $\langle \sigma, \tau \rangle$.

The proof of the two previous lemmas is an easy adaptation of the proofs of their counterpart for P-i.j. strategies.

Lemma 6.59. *Let $\tau : I \to C_2$, $\sigma : C_1 \otimes C_2 \to B$ and $\mathfrak{M}$ be any set of moves initial in $C_1 \otimes C_2$. If τ is P-i.j. and σ is P-i.j. (resp. closed P-i.j.) modulo $\mathfrak{M}$ then $(id_{C_1} \otimes \tau) \fatsemi \sigma$ is P-i.j. (resp. closed P-i.j.) modulo $\mathfrak{M} \cap C_1$.*

Proof. Let $D = C_1 \otimes C_2$. Let $u \in Int(C_1, D, B)$ be a non-empty interaction play of $\mu = (id_{C_1} \otimes \tau)^\dagger \| \sigma$, and m denote the last play of u. We need to show that if m does not belong to $\mathfrak{M}$ then $u \upharpoonright C_1, B$ is P-incrementally justified.

Suppose $m \in C_1 \setminus \mathfrak{M}$. Let d be the initial D-move hereditarily justifying m, then by definition of μ we have $u \upharpoonright C_1, D, d \in id_{C_1}$ which implies that $u \upharpoonright C_1, B = u \upharpoonright D, B$. But u is an interaction sequence therefore $u \upharpoonright D, B \in \sigma$, and since σ is P-i.j. modulo $\mathfrak{M}$ this implies that $u \upharpoonright C_1, B$ is P-incrementally justified.

Suppose $m \in B$ then necessarily its justifier also occurs in B. By definition of u, the play $u \upharpoonright D, B$ belongs to σ which is P-i.j. modulo $\mathfrak{M}$. Since m belongs to B it cannot be in $\mathfrak{M}$ therefore u is P-i.j. Furthermore, since τ is P-i.j., so is $(id_{C_1} \otimes \tau)^\dagger$ therefore the play $u \upharpoonright C_1, D$ and all its prefixes are P-i.j. Hence we can apply Lemma 6.43 with $X \leftarrow D, B$ and $Y \leftarrow C_1, D$ which shows that $u \upharpoonright C_1, B$ is P-i.j. ☐

Lemma 6.60. *Let $mkvar : B \to C$ be the denotation of the* `mkvar` *construct where $B = (\mathtt{exp}^1 \to \mathtt{com}) \times \mathtt{exp}$ and $C = \mathtt{var}$. If $\sigma : A \to B$ is a closed P-i.j. strategy modulo $\mathfrak{M}_A \cup [\![\mathtt{exp}^1]\!]$ for some set $\mathfrak{M}_A$ of initial A-moves then $\sigma; mkvar$ is closed P-i.j. modulo $\mathfrak{M}_A$.*

Proof. Let u be an interaction sequence such that $u \upharpoonright A, C$ ends with a P-question that is not in $\mathfrak{M}_A$. Then $u \upharpoonright A, B$ and $u \upharpoonright B, C$ are both P-i.j. Let m denote the last move in u and n be its justifier in $u \upharpoonright A, C$. Suppose that an O-move p occurs in the P-view between n and m. We show that its order is necessarily smaller than that of m. We necessarily have $m \in \circ_A$ because there is no P-question in C.

(a) Suppose that $m \in \circ_A$, $n \in \bullet_A$ and $p \in \bullet_A$. In general, p does not necessarily appear in the P-view $\ulcorner u \upharpoonright A, B \urcorner$ (see proof of compositionality). In the present case, however, this case never happens. Indeed, as noted in the proof of Lemma 6.43, this would imply that p lies underneath a $◐$- $◑$-arc. But this is not possible since the only $◐$-move in B is an initial move. Thus p occurs in $\ulcorner u \upharpoonright A, B \urcorner$ and since $u \upharpoonright A, B$ is P-i.j. this implies that p has order smaller than m.

(b) Suppose that $m \in \circ_A$, $n \in \bullet_A$ and $p \in \bullet_C$. Take $Y = A, B$ and $X = B, C$. We have that $u \upharpoonright Y$ is P-i.j. and since *mkvar* is a P-i.j. strategy, for all B-move b occurring in u, $u_{\leqslant b} \upharpoonright X$ is P-i.j. Thus we can apply Lemma 6.40 which shows that $\mathrm{ord}_{A \to C}\, p \leq \mathrm{ord}_{A \to C}\, m$.

(c) Suppose $m \in \circ_A$, $n \in \bullet_C$. Then in A, B, the move m is justified by a $◐$-move b_0 itself justified by n in B, C. By definition of the strategy *mkvar*, n and b_0 are in fact consecutive moves in u, thus p necessarily occurs after b_0. If $p \in \bullet_C$ then we conclude with Lemma 6.40 as in (b) that $\mathrm{ord}_{A \to C}\, p \leq \mathrm{ord}_{A \to C}\, m$. Otherwise $p \in \bullet_A$, and we conclude as in (a).

Hence $u \upharpoonright A, C$ is P-i.j. It is further *closed* P-i.j. because both $u \upharpoonright A, B$ and $u \upharpoonright B, C$ are. ☐

Example 6.61. The unsafe term

$$f : (\mathtt{exp} \to \mathtt{exp}) \to \mathtt{com} \vdash \lambda x^{\mathtt{exp}}.f(\lambda y^{\mathtt{exp}}.\underline{x}) \equiv M : \mathtt{exp}^1 \to \mathtt{com}$$

is denoted by a strategy $[\![M]\!]$ that is closed P-i.j. modulo $[\![\mathtt{exp}^1]\!]$. But the term $\mathtt{mkvar}\, M\, 0 : \mathtt{var}$ is denoted by the strategy $\langle [\![M]\!], 0\rangle; mkvar$ which is closed P-i.j.

Given a safe split-term $\Gamma|\Xi \vdash_s M : A$, we write $[\![\Gamma|\Xi \vdash_s M : A]\!]$ to refer to $[\![\Gamma, \Xi \vdash M : A]\!]$, the game denotation of the corresponding IA split-term. For every game A we write $In(A)$ for the set of initial moves in A.

Proposition 6.62. *Let $\Gamma|\Xi \vdash_s M : A$ be a safe IA split-terms. Its denotation $[\![\Gamma|\Xi \vdash_s M : A]\!]$ is closed P-i.j. modulo $In([\![\Xi]\!])$.*

Remark 6.63 $In([\![\Xi]\!])$ contains only order-0 questions because the context Ξ contains variables of type `var` and `exp` only.

Proof. We only need to prove the result for terms where the only allowed uses of the Y combinator is in subterms of the form Ω; the result then follows immediately using the syntactic approximants technique and continuity of the "closed P-i.j." property.

We proceed by induction on the safe IA term. The cases (var), (wk), (const), (succ), (pred), (cond) are the same as for safe PCF.

- $(\mathsf{var}^{\mathtt{new}})$, $(\mathsf{wk}^{\mathtt{new}})$ are similar to (var) and (wk).

- (seq), (assign), (deref) These constants all have closed P-i.j. denotations so the result follows from the I.H., Lemma 6.56, Proposition 6.58 and 6.57.

- (app) The premise of the rule is an almost safe split-term: it is a consecutive applications of safe terms. By the I.H. each of these terms has a denotation that is closed P-i.j. modulo $In([\![\Xi]\!])$. Since the evaluation strategy *ev* is P-i.j., by Lemma 6.56, the denotation of the split-term being formed is P-i.j. modulo $In([\![\Xi]\!])$. Finally, the side-condition of the rule ensures that it is *closed* P-i.j. modulo $In([\![\Xi]\!])$.

- (abs) It follows from the I.H. and because the side-condition of the abstraction rules constrains only free variables from the Γ-context.

- (new) Let $\sigma = [\![\Gamma|\Xi, x : \mathtt{var} \vdash_s M : B]\!]$. We have $[\![\Gamma|\Xi \vdash_s \mathtt{new}\ x\ \mathtt{in}\ M : B]\!] = (id_{\Gamma,\Xi} \otimes cell) ⨾ \sigma$ where *cell* denotes the memory cell strategy on the game $I \rightarrow !\mathtt{var}$. By the I.H. σ is closed P-i.j. modulo $In([\![\Xi \otimes !\mathtt{var}]\!])$. Instancing Lemma 6.59 with $\tau \leftarrow cell$, $C_1 \leftarrow \Gamma \otimes \Xi$ and $C_2 \leftarrow !\mathtt{var}$ gives us the desired result.

- (mkvar) Let $\sigma = [\![\Gamma|\Xi \vdash_s \mathtt{mkvar}\ (\lambda x.M_1)M_2]\!]$. We have $\sigma = \langle \Delta(\sigma_1), \sigma_2\rangle; mkvar$ where $\sigma_1 = [\![\Gamma|\Xi, x : \mathtt{exp} \vdash_s M_1 : \mathtt{com}]\!]$ and $\sigma_2 = [\![\Gamma|\Xi \vdash_s M_2 : \mathtt{exp}]\!]$. By the I.H., σ_1 is closed P-i.j. modulo $In([\![\Xi, x : \mathtt{exp}]\!])$ and σ_2 is closed P-i.j. modulo $In([\![\Xi]\!])$ therefore since $\operatorname{ord}\Gamma \geq 1$ (by the side-condition of the (mkvar) rule) the strategy $\langle \Delta(\sigma_1), \sigma_2\rangle : [\![\Gamma \times \Xi \rightarrow (\mathtt{exp}^1 \rightarrow \mathtt{com}) \times \mathtt{exp}]\!]$ is closed P-i.j. modulo $In([\![\Xi]\!] \cup [\![\mathtt{exp}^1]\!])$. Hence by Lemma 6.60, σ is closed P-i.j. modulo $In([\![\Xi]\!])$. $\square$

By definition, safe IA terms are the semi-closed safe split-terms, hence:

Corollary 6.64. *In the standard game model of IA, safe terms are denoted by closed P-i.j. strategies.*

6.4 O-incremental justification

We define *O-incremental justification* as the dual of P-incremental justification:

Definition 6.65.
(i) A play sm of *odd* length is said to be ***O-incrementally justified***, or *O-i.j.* for short, if m points to the last unanswered P-question in $\ulcorner s \urcorner$ with order strictly greater than $\operatorname{ord} m$.
(ii) A strategy σ is said to be ***O-incrementally justified***, if all plays in σ ending with an O-question are O-incrementally justified.

Think of O-incremental justification as the constraint that one needs to impose to reflect the fact that the environment is incarnated by a safe term. The duality between O-i.j. and P-i.j. is similar to that of O-visibility versus P-visibility [Har05, Sec. 3.6].

For every strategy σ, we write $\mathcal{O}(\sigma)$ to denote the largest subset of plays of σ whose odd-length prefixes are all O-i.j. The set $\mathcal{O}(\sigma)$ is obtained by removing all the plays containing O-moves that are not incrementally justified. It defines a strategy that mimics the strategy σ as long as the Opponent plays incrementally and does not answer otherwise.

Lemma 6.66. *Let $\sigma : A$ and $\alpha : A \rightarrow o$ be two strategies. Then in the composition $\sigma;\alpha$, the P-i.j. plays of σ interact only with O-i.j. plays of α, and the O-i.j. plays of σ interact only with P-i.j. plays of α.*

Proof. Let $\sigma : A$ and $\alpha : A \rightarrow o$ be two strategies, and q be the initial move of the game $[\![A \rightarrow o]\!]$. For every $s \in L_A$ we have $qs \in L_{A\rightarrow o}$. P-moves and O-moves in $[\![A]\!]$ become O-moves and P-moves in $[\![A \rightarrow o]\!]$ respectively. Hence P-views of plays in A correspond to O-views in $A \rightarrow o$; thus $q\ulcorner s \urcorner^A = \llcorner qs \lrcorner_{A\rightarrow o}$. Now take an interaction sequence $u = qv \in \sigma \| \alpha$. We have $u \upharpoonright (A \rightarrow o) = (qv) \upharpoonright (A \rightarrow o) = q(v \upharpoonright A)$. Hence if $u \upharpoonright A = v \upharpoonright A$ is P-i.j. then by the previous remark, $u \upharpoonright (A \rightarrow o)$ is O-i.j. The proof of the second part is symmetrical. $\square$

Lemma 6.67. *In an order*-3 well-opened *game all the legal positions are O-i.j.*

Proof. Let A be an order-3 well-opened game. Take a play s in σ ending with a question move q. We prove by induction on s that if q is a non-initial O-move then there is a single P-move in $\llcorner s \lrcorner$ with order $> \operatorname{ord} q$ (and thus s is necessarily O-i.j.). We do a case analysis on the level of q. We recall that $\operatorname{ord} q + \operatorname{level} q \leq \operatorname{ord} A$. Since q is a non-initial O-move, we necessarily have $\operatorname{level} q = 2$. Let q' denote the P-move preceding q in s. Suppose that $\operatorname{level} q' = 1$ then q' is justified by an occurrence of the initial A-move q_0. Since A is well-opened, s contains only one occurrence of q_0 and therefore we have $\llcorner s \lrcorner = q_0 \cdot q' \cdot q$. Thus by O-visibility, q necessarily points to q' therefore $\operatorname{ord} q' > \operatorname{ord} q$; thus since q' is the only P-move occurring in the O-view, it is also the only P-move with order greater than $\operatorname{ord} q$. Otherwise we have $\operatorname{level} q' = 3$. Thus $\operatorname{ord} q' \leq \operatorname{ord} A - \operatorname{level} q' = 0$ and q' is justified by some O-move q'' of level 2. We have $\llcorner s \lrcorner = \llcorner s_{\leqslant q''} \lrcorner \cdot q' \cdot q$ so we can conclude using the I.H. on $s_{\leqslant q''}$ and the fact that $\operatorname{ord} q' = 0 < \operatorname{ord} q$. ☐

This lemma does not hold anymore at order 4. For instance the identity strategy $id_A : A \to A$ on the order-3 game $A = [\![((o^3 \to o^2) \to o^1) \to o^0]\!]$ contains the following play which is not O-i.j.:

$$q_0\ q'_0\ q'_1\ q_1\ q_2\ q'_2\ q'_1\ q_1\ q_2\ q'_2\ \underline{q'_3}$$

where primed moves correspond to moves from the left copy of A.

Corollary 6.68. *Let σ, μ be two strategies from $\mathcal{C}(I, A)$ where A is an order-3 game. Then*

$$\sigma \lesssim \mu \iff \sigma \lesssim_{\mathcal{I}} \mu\ .$$

Proof. Let $\alpha : A \to o$ be a test strategy. By Lemma 6.67, σ and μ are necessarily O-i.j. Thus by Lemma 6.66, the plays of σ, μ can only interact with P-i.j. plays from α. Hence $\sigma;\alpha = \sigma;\mathcal{P}(\alpha)$ and $\mu;\alpha = \mu;\mathcal{P}(\alpha)$. Therefore by definition of the intrinsic preorders we have $\sigma \lesssim \mu$ iff $\sigma \lesssim_{\mathcal{I}} \mu$. ☐

6.5 Full abstraction

Question: What is a fully abstract model of safe PCF and safe IA?

We already know from the fully-abstract game model of PCF that when the observational preorder is defined with respect to unrestricted (*i.e.*, possibly unsafe) PCF contexts, observational equivalence is captured by equality of the quotiented game denotations. We show here that a similar correspondence holds when observational equivalence is defined with respect to safe contexts only. This further implies a full abstraction result for the fragments of PCF and IA consisting of safe closed terms.

Observational equivalence with respect to safe contexts

We first recall some basic definitions. A *context* is a PCF term containing exactly one free occurrence of a distinguished variable '$-$' called the "hole". A context is usually denoted by $C[-]$ so that

$$- : A \vdash C[-] : B$$

is a valid PCF term-in-context for some type A and B. For every term M of type A we write $C[M]$ to denote the term obtained by substituting M for the hole using capture-permitting substitution. Due to the possibility of variable capture, this term is not necessarily a valid PCF term. Also it is possible to have $C_1[-] =_\beta C_2[-]$ and $C_1[M] \neq_\beta C_2[M]$. (For instance take $C_1[-] \equiv (\lambda x^{\mathsf{exp}}.-)0$ and $C_2[-] \equiv (\lambda x^{\mathsf{exp}}.-)\Omega$. Then $C_1[-] =_\beta - =_\beta C_2[-]$; but $C_1[x] =_\beta 0$ and $C_2[x] =_\beta \Omega$.)

We write $\mathsf{Trm}(\Gamma, A)$ for the set of terms M such that $\Gamma \vdash M : A$ is derivable in PCF. Terms in $\mathsf{Trm}(\emptyset, \mathsf{exp})$ (*i.e.*, closed PCF terms of base type) are called ***PCF program***. For every typing-context Γ and type $A \in \mathbb{T}$ the ***program contexts*** $\mathsf{Ctxt}(\Gamma, A)$ are the PCF contexts $C[-]$ such that for all $M \in \mathsf{Trm}(\Gamma, A)$, the term $C[M]$ is a PCF program.

We write $\mathsf{Trm_s}(\Gamma, A)$ for the set of terms M such that $\Gamma \vdash_{\mathsf{s}} M : A$ is derivable in safe PCF. We say that a PCF context $C[-]$ is a ***safe context*** if the judgement

$$- : A \vdash_{\mathsf{s}} C[-] : B,$$

is a valid safe PCF term-in-context. The ***safe program contexts*** $\mathsf{Ctxt_s}(\Gamma, A)$ are the program contexts from $\mathsf{Ctxt}(\Gamma, A)$ that are safe contexts.

We now define two notions of observational preorder for PCF:

Definition 6.69 (Observational preorders)**.** Let Γ be a typing-context and T be a simple type. Let M and N range over $\mathsf{Trm}(\Gamma, T)$. We write $\lesssim$ to denote the standard observational preorder for PCF terms. This relation on $\mathsf{Trm}(\Gamma, T)$ is defined as:

$$M \lesssim N \stackrel{\text{def}}{=} \forall C[-] \in \mathsf{Ctxt}(\Gamma, A).\ C[M] \Downarrow \implies C[N] \Downarrow \ .$$

The relation $\lesssim_s$ on $\mathsf{Trm}(\Gamma, T)$ is defined similarly to $\lesssim$ except that contexts range over safe terms only:

$$M \lesssim_s N \stackrel{\text{def}}{=} \forall C[-] \in \mathsf{Ctxt_s}(\Gamma, A).\ C[M] \Downarrow \implies C[N] \Downarrow \ .$$

We write $\cong$ and $\cong_s$ to denote the reflexive closures of $\lesssim$ and $\lesssim_s$.

Lemma 6.70.

(i) The relations $\lesssim$ and $\lesssim_s$ are preorders (reflexive and transitive);

(ii) Consequently $\cong$ and $\cong_s$ are equivalence relations;

(iii) $\lesssim \subseteq \lesssim_s$.

Proof. Trivial. □

Note that in the definition of $\lesssim_s$, the program context $C[-]$ ranges in $\mathsf{Ctxt_s}(\Gamma, A)$ but it is not required that $C[M]$ and $C[N]$ are themselves safe. When restricted to safe terms, however, $C[M]$ and $C[N]$ are necessarily safe:

Lemma 6.71. $M \in \mathsf{Trm_s}(\Gamma, T) \wedge C[-] \in \mathsf{Ctxt_s}(\Gamma, T) \implies C[M] \in \mathsf{Trm_s}(\emptyset, \mathtt{exp})$.

Proof. Suppose that $M \in \mathsf{Trm_s}(\Gamma, T)$ and $C[-] \in \mathsf{Ctxt_s}(\Gamma, T)$ then in particular, $M \in \mathsf{Trm}(\Gamma, T)$ and $C[-] \in \mathsf{Ctxt}(\Gamma, T)$, therefore by definition of a program context we have $C[M] \in \mathsf{Trm}(\emptyset, \mathtt{exp})$.

Plugging a term in the context is done via capture-permitting substitution: $C[M]$ is given by $(C[-])\,\{M/-\}$. But since both $C[-]$ and M are safe and $C[M]$ is a valid term, by the No-variable-capture lemma (Corollary 3.85(ii)) it is syntactically equivalent to perform the standard substitution: $C[M] \equiv (C[-])\,[M/-]$. Hence by the Substitution Lemma 3.19, $C[M]$ is safe. □

Lemma 6.72. $M \in \mathsf{Trm_s}(\Gamma, T) \wedge C[-] \in \mathsf{Ctxt_s}(\Gamma, T) \implies [\![C[M]]\!] = [\![M]\!];[\![C[-]]\!]$.

Proof. By the previous lemma, plugging M in $C[-]$ can be done using the capture-permitting substitution therefore $[\![C[M]]\!] = [\![M]\!];[\![C[-]]\!]$. □

Note that this lemma does not hold for unsafe context. For instance with $C[-] \equiv (\lambda x^{\mathtt{exp}}.-)\Omega$ we have $[\![x]\!];[\![C[-]]\!] = [\![x]\!];id_A = [\![x]\!]$ but $[\![C[x]]\!] = \bot$.

Remark 6.73 It is possible to define a third notion of observational preorder where the contexts are unrestricted but where we require instead that $C[M]$ and $C[N]$ are safe. This notion of observational preorder differs from $\lesssim_s$ because the safety of $C[M]$ does not necessarily implies that of $C[-]$ (*e.g.*, the context $- : A \vdash \lambda x^A.- : B$ is unsafe although $C[x]$ is safe).

Remark 6.74 Compared to $\lesssim$, the observational preorder $\lesssim_s$ is a relatively coarse approximation relation for open terms. If we fix a type T then all the open terms of type T containing variables of order at least T will be equated by $\lesssim_s$. The is because for every such term M, there is no safe context $C[-]$ such that $C[M]$ is closed. Indeed, if $C[M]$ is closed then all the free variables in M must be abstracted in $C[M]$. Take a variable $z \in FV(M)$ satisfying $\operatorname{ord} z \geq \operatorname{ord} T$, then the hole in $C[-]$ must appear in a subterm of the form $\lambda z. \cdots - \cdots$ containing the hole '–'. But then this implies that the context is unsafe because the hole, which has order smaller than z, is not abstracted with z. For example, the terms $x : \mathtt{exp} \vdash \mathtt{cond}\, 0\, x\, i \equiv M_i : \mathtt{exp}$ for $i \in \mathbb{N}$ are all $\cong_s$-equivalent, but their closures $N_i \equiv \lambda x^{\mathtt{exp}}.M_i$ are not: $N_i \not\lesssim_s N_j$ for every $i \neq j$.

Proposition 6.75 (Computational Adequacy)**.** *Let P be a PCF program. Then*

$$P \Downarrow \iff [\![P]\!]_{\mathcal{C}} \neq \bot \iff [\![P]\!]_{\mathcal{C}} \not\lesssim_{\mathcal{I}} \bot$$

where $\bot$ denotes the empty strategy on the game $[\![\mathtt{exp}]\!]$.

Proof. The first equivalence is the Computational Adequacy result for PCF [AM97]. The second equivalence follows from the fact that the $\lesssim_{\mathcal{I}_{ib}}$-equivalence class of $\bot$ contains only the strategy $\bot$ itself. Indeed, suppose that $\sigma \lesssim_{\mathcal{I}_{ib}} \bot$. Let Σ denote the Sierpinski game and $\bot_\Sigma$ denote the empty strategy on Σ. Take $\alpha : [\![\mathtt{exp}]\!] \rightarrow \Sigma$ to be the following P-i.j. strategy

$$\alpha = \{\epsilon, q \cdot q'\} \cup \{q \cdot q' \cdot i \cdot a \mid i \in \mathbb{N}\}$$

where q' denotes the initial move of $[\![\mathtt{exp}]\!]$.

We have $\bot \fatsemi \alpha = \bot_\Sigma$ therefore since $\sigma \lesssim_{\mathcal{I}_{ib}} \bot$ we must have $\sigma \fatsemi \alpha = \bot_\Sigma$. There are only two kinds of strategy on the game $[\![\mathtt{exp}]\!]$: the empty strategy $\bot$ and $\{\epsilon, q \cdot i\}$ for $i \in \mathbb{N}$. Only the empty strategy verifies the equation $\sigma \fatsemi \alpha = \bot_\Sigma$ hence $\sigma = \bot$. $\square$

Proposition 6.76 (Inequational soundness)**.** *Let $M, N \in \mathsf{Trm}(\Gamma, T)$. Then:*

$$[\![M]\!]_{\mathcal{C}} \subseteq [\![N]\!]_{\mathcal{C}} \implies M \mathrel{\underset{\sim}{\sqsubset}}_s N \ .$$

Proof. It follows from Inequational soundness in $\mathcal{C}$ [AM97] since $\mathrel{\underset{\sim}{\sqsubset}}$ is a subset of $\mathrel{\underset{\sim}{\sqsubset}}_s$. $\square$

Theorem 6.77 (Inequational soundness in $\mathcal{C}_{ib}/\lesssim_{\mathcal{I}_{ib}}$)**.** *Let $M, N \in \mathsf{Trm}(\Gamma, T)$. Then:*

$$[\![M]\!]_{\mathcal{C}} \lesssim_{\mathcal{I}_{ib}} [\![N]\!]_{\mathcal{C}} \implies M \mathrel{\underset{\sim}{\sqsubset}}_s N \ .$$

Proof. We first show the result for closed terms. We follow the same argument as the proof of Inequational soundness for PCF [AM97]. Let $M, N \in \mathsf{Trm}(\emptyset, T)$ and suppose that $[\![M]\!]_{\mathcal{C}} \lesssim_{\mathcal{I}_{ib}} [\![N]\!]_{\mathcal{C}}$ and that $C[M] \Downarrow$ for some safe context $C[-]$. Then the denotation of $C[-]$ is a P-i.j. strategy $\alpha \in \mathcal{I}(T, \Sigma)$. For every closed term P, the context-substitution $C[P]$ causes no variable capture therefore we have $[\![C[P]]\!] = [\![P]\!] \fatsemi \alpha$. Thus by Computational Adequacy we have $[\![M]\!] \fatsemi \alpha \neq \bot$. But since $[\![M]\!]_{\mathcal{C}} \lesssim_{\mathcal{I}} [\![N]\!]_{\mathcal{C}}$ this implies that $[\![N]\!] \fatsemi \alpha \neq \bot$ which by Computational Adequacy implies $C[N] \Downarrow$ as required.

We now generalize the result to open terms. We first observe that for all $C[-] \in \mathsf{Ctxt_s}(\Gamma, T)$ and $M \in \mathsf{Trm}(\Gamma, T)$ where $\Gamma = \overline{x} : \overline{A}$ we have:

$$C[M] \Downarrow \iff C[\lambda \overline{x}^{\overline{A}}.M\overline{x}] \Downarrow \iff C'[\lambda \overline{x}^{\overline{A}}.M] \Downarrow \tag{6.4}$$

where $C'[-]$ is the program context defined as $C'[-] \equiv C[-\,\overline{x}]$. It is easy to see that this context is necessarily safe: $C'[-] \in \mathsf{Ctxt_s}(\Gamma, (\overline{A}, T))$.

Now take two open terms $M, N \in \mathsf{Trm}(\Gamma, T)$ where $\Gamma = \overline{x} : \overline{A}$. W.l.o.g. we can assume that the list $\overline{x}$ contains exactly the variables from $FV(M) \cup FV(N)$. We have

$$\begin{aligned}
[\![M]\!]_{\mathcal{C}} \lesssim_{\mathcal{I}_{ib}} [\![N]\!]_{\mathcal{C}} &\iff \Lambda^{|\overline{x}|}([\![\lambda \overline{x}^{\overline{A}}.M]\!]_{\mathcal{C}}) \lesssim_{\mathcal{I}_{ib}} \Lambda^{|\overline{y}|}([\![\lambda \overline{x}^{\overline{A}}.N]\!]_{\mathcal{C}}) \\
&\iff [\![\lambda \overline{x}^{\overline{A}}.M]\!]_{\mathcal{C}} \lesssim_{\mathcal{I}_{ib}} [\![\lambda \overline{x}^{\overline{A}}.N]\!]_{\mathcal{C}} \\
&\iff \forall C'[-] \in \mathsf{Ctxt_s}(\Gamma, (\overline{A}, T)).C'[\lambda \overline{x}^{\overline{A}}.M] \Downarrow \implies C'[\lambda \overline{x}^{\overline{A}}.N] \Downarrow \\
&\iff \forall C[-] \in \mathsf{Ctxt_s}(\Gamma, T).C[M] \Downarrow \implies C[N] \Downarrow && \text{by (6.4)} \\
&\implies M \mathrel{\underset{\sim}{\sqsubset}}_s N \ .
\end{aligned}$$

$\square$

The ***star fragment*** of PCF written PCF^*, consists of all the judgements $\Gamma \vdash M : T$ satisfying the condition:

$$\forall z : A \in \Gamma. \operatorname{ord} A < \operatorname{ord} T \tag{6.5}$$

abbreviated as "$\operatorname{ord} \Gamma < \operatorname{ord} T$".

Theorem 6.78 (Full abstraction of PCF^* with respect to safe contexts)**.** *Let* $M, N \in \mathsf{Trm}(\Gamma, T)$ *be two PCF terms with* $\operatorname{ord}\Gamma < \operatorname{ord} T$*. Then*

$$\begin{aligned} M \sqsubseteq_s N &\iff [\![M]\!]_{\mathcal{C}} \lesssim_{\mathcal{I}_{ib}} [\![N]\!]_{\mathcal{C}} & (i)\\ &\iff \mathcal{O}([\![M]\!]_{\mathcal{C}}) \lesssim_{\mathcal{I}_{ib}} \mathcal{O}([\![N]\!]_{\mathcal{C}}) \ . & (ii)\end{aligned}$$

Proof. (i) ($\Leftarrow$) This is the Inequational Soundness result (Theorem 6.77). ($\Rightarrow$) We follow the same argument as the proof of Full Abstraction of PCF [AM97]. Suppose that $[\![M]\!]_{\mathcal{C}} \lesssim_{\mathcal{I}_{ib}} [\![N]\!]_{\mathcal{C}}$. Then by definition of the preorder $\lesssim_{\mathcal{I}_{ib}}$, there exists a P-i.j. strategy $\alpha : (\Gamma \to [\![T]\!]) \to \mathtt{exp}$ such that $[\![M]\!] \fatsemi \alpha = \top$ and $[\![N]\!] \fatsemi \alpha = \bot$. The strategy α can be chosen to be compact, and since $\operatorname{ord}(T) \geq \operatorname{ord}(\mathtt{exp}) = 0$, it is closed P-i.j. By the definability result for safe PCF (Prop. 5.27), there exists a closed safe term-in-context $\vdash_{\mathsf{s}} \lambda z^{\Gamma\to T}.Q : (\Gamma \to T) \to \mathtt{exp}$ such that $[\![\lambda z^{\Gamma\to T}.Q]\!] = \alpha$. Using the application rule and the abstraction we can then form the safe program context: $- : T \vdash_s (\lambda z^{\Gamma\to T}.Q)(\lambda y^{\Gamma}.-) \equiv C[-] : \mathtt{exp}$. In particular, the subterm $\lambda y^{\Gamma}.-$ is safe because we have $\operatorname{ord} - = \operatorname{ord} T > \operatorname{ord}\Gamma$ by assumption. Clearly, $[\![C[-]]\!] \cong [\![\lambda z^{T}.Q]\!] = \alpha$ therefore by Computational Adequacy it follows that $C[M] \Downarrow$ and $C[M] \not\Downarrow$.

(ii) In the definition of the preorder $\lesssim_{\mathcal{I}_{ib}}$, the test strategy α ranges over P-i.j. strategies therefore by Lemma 6.66 α can only interact with O-i.j. plays. Hence for every strategy σ in $\mathcal{C}$, $\mathcal{O}(\sigma)$ and σ are in the same $\lesssim_{\mathcal{I}_{ib}}$-equivalence class. ☐

Full abstraction of safe PCF

Although the small-step operational semantics of PCF and safe PCF differ—the former contracts β-redexes one at a time whereas the latter contracts "consecutive" β-redexes in a single step—they have the same big-step operational semantics: a safe term evaluates to a value in safe PCF if and only if it evaluates to the same value in PCF. Hence the operational semantics of safe PCF is given by the same relation $\Downarrow$ as PCF.

We now consider the restrictions of the relations $\sqsubseteq$ and $\sqsubseteq_s$ on $\mathsf{Trm}(\Gamma, T) \times \mathsf{Trm}(\Gamma, T)$ to $\mathsf{Trm_s}(\Gamma, T) \times \mathsf{Trm_s}(\Gamma, T)$. Clearly these restrictions define preorders on $\mathsf{Trm_s}(\Gamma, T)$.

Theorem 6.79 (Full abstraction for closed safe PCF terms)**.** *Let* M, N *be two* closed *safe PCF terms of the same type. Then*

$$\begin{aligned} M \sqsubseteq_s N &\iff [\![M]\!]_{\mathcal{I}} \lesssim_{\mathcal{I}_{ib}} [\![N]\!]_{\mathcal{I}}\\ &\iff \mathcal{O}([\![M]\!]_{\mathcal{I}}) \lesssim_{\mathcal{I}_{ib}} \mathcal{O}([\![N]\!]_{\mathcal{I}}) \ .\end{aligned}$$

Proof. Safe closed PCF terms are all in PCF^* therefore the result follows immediately from Theorem 6.78 since for every safe term M we have $[\![M]\!]_{\mathcal{I}} = [\![M]\!]_{\mathcal{C}}$. ☐

Remark 6.80 Observe that the condition (6.5) used in Theorem 6.78 expresses precisely the negation of the basic property of the safe lambda calculus. Therefore the star fragment of safe PCF is precisely given by the set of *closed* safe terms. That is why our full abstraction result holds only for the fragment of PCF consisting of *closed* terms.

Full abstraction fails for open terms. For instance the family of opened safe terms $\mathtt{cond}\,0\,x\,i$ for $i \in \mathbb{N}$ are all in the same $\sqsubseteq_s$-equivalence class although their denotations are not in the same $\lesssim_{\mathcal{I}_{ib}}$-equivalence class.

In fact the observational relation $\sqsubseteq_s$ trivially equates all open safe terms of a given type! This is due to the fact that for every open safe term M, there is no safe context $C[-]$ such that the term $C[M]$ is closed. (See remark 6.74.)

Full abstraction of Safe Idealized Algol

The proof of full abstraction of Idealized Algol is based on the Innocent Factorization theorem:

Theorem 6.81 (Innocent Factorization [AM97])**.** *For every strategy σ on an IA game A, there exists an innocent strategy $\tau : !\mathtt{var} \multimap A$ such that $\sigma = cell;\tau$.*

A version of this theorem also holds for safe IA:

Theorem 6.82 (Innocent P-i.j. Factorization)**.** *For every closed P-i.j. strategy σ on an IA game A, there is an innocent strategy $\mu : !\mathtt{var} \multimap A$ which is closed P-i.j. modulo $In([\![!\mathtt{var}]\!])$ and such that $\sigma = cell;\mu$.*

Proof. By the Factorization Theorem we have that $\sigma = cell;\tau$ for some innocent strategy $\tau : !\mathtt{var} \multimap A$. Observe that τ is not necessarily P-i.j. modulo $In([\![!\mathtt{var}]\!])$, although σ is P-i.j. (see the following remark). However all the plays of τ interacting with *cell* are P-i.j. modulo $In([\![!\mathtt{var}]\!])$. Indeed, take an interaction play $u \in Int(I, !\mathtt{var}, A)$ ending with an A-move. It is easy to see that P-view of the play $u \upharpoonright I, A$ is obtained from the P-view of the play $u \upharpoonright !\mathtt{var}, A$ by removing the moves played in $[\![!\mathtt{var}]\!]$. Thus because the question moves of the game $[\![!\mathtt{var}]\!]$ are of order 0, since $u \upharpoonright I, A$ is P-i.j. so must be $u \upharpoonright !\mathtt{var}, A$.

Take μ to be the strategy obtained by truncating all the plays in τ that are not P-i.j. Then clearly μ is P-i.j. modulo $In([\![!\mathtt{var}]\!])$ and satisfies $\sigma = cell;\mu$. ☐

Remark 6.83 In the previous proof, we mentioned that it is possible for $cell;\tau$ to be P-i.j even when τ is not P-i.j. modulo $In([\![!\mathtt{var}]\!])$. Here is an example illustrating this fact. The IA term

$$x : \mathtt{var} \vdash \lambda f^2\, y^{\mathtt{exp}}.\mathtt{seq}\,(\mathtt{assign}\, x\, 0)\ (\mathtt{cond}\,(\mathtt{deref}\, x)\ 0\ (f(\lambda z^{\mathtt{exp}}.\underline{y}))) \equiv M$$
$$: \mathtt{var}^0 \to ((\mathtt{exp}^1 \to \mathtt{exp}^2) \to \mathtt{exp}^3) \to \mathtt{exp}^4 \to \mathtt{exp}^5$$

is unsafe because it contains the unsafe occurrence y, and its denotation is not P-i.j. modulo $In([\![!\mathtt{var}]\!])$ because it contains the play:

$$q_5\ \mathtt{write}_0\ \mathtt{done}\ \mathtt{read}\ 1\ q_3\ q_2\ \underline{q_4}\ .$$

The term $\mathtt{new}\ x\ \mathtt{in}\ M$, however, is observationally equivalent to 0 and therefore its denotation is P-i.j.

Using this factorization result we can then show definability for safe IA:

Theorem 6.84 (Definability)**.** *Let T be some IA type.*

(i) Every innocent compact strategy $\sigma : [\![!\mathtt{var}]\!] \to [\![T]\!]$ that is closed P-i.j. modulo $In([\![!\mathtt{var}]\!])$ is definable by some safe IA split-term $\emptyset | v : \mathtt{var} \vdash_{\mathsf{s}} M : T$;

(ii) Every compact closed P-i.j. strategy $\sigma : I \to [\![T]\!]$ is definable in safe IA.

Proof. (i) By the *Definability for Normed Sequential Categories* theorem by Abramsky and McCusker [AM97], the strategy σ is definable by some IA term $v : \mathtt{var} \vdash M : T$. By construction, M is in beta-eta normal form, and there is a correspondence between its variable occurrences and P-questions in game semantics so that a play ending with a P-question is closed P-i.j. just if the corresponding variable is incrementally bound (See the argument based on the Correspondence Theorem used in Chapter 5). This corresponds precisely to the restriction imposed by the side-conditions in the typing rules of safe IA. Plays ending with an initial move in $[\![!\mathtt{var}]\!]$ are not necessarily closed P-i.j., therefore the corresponding variable is not necessarily incrementally bound. But this is permitted since the side-condition of the typing rules constrain only variables from the Γ-context. Hence we can type the judgement $\emptyset | v : \mathtt{var} \vdash_{\mathsf{s}} M : T$.

(ii) By the Factorization Theorem we have $\sigma = cell;\mu$ for some innocent strategy $\mu : !\mathtt{var} \multimap [\![T]\!]$ that is closed P-i.j. modulo $In([\![!\mathtt{var}]\!])$, and by (i) μ is the denotation of some safe IA split-term $\emptyset | v : \mathtt{var} \vdash_{\mathsf{s}} M : (\overline{A}, B)$ for some types $\overline{A} = A_1, \ldots, A_n$ and base type B. We can then conclude as in the case of IA. For instance for $B = \mathtt{exp}$ we have $\sigma = [\![\lambda \overline{x}^{\overline{A}}.\mathtt{new}\ v\ \mathtt{in}\ M\overline{x}]\!]$. ☐

Inequational Soundness of the game model follows from that of IA. We then obtain:

Theorem 6.85 (Full abstraction for closed safe IA terms). *Let $\vdash_s M : T$ and $\vdash_s N : T$ be two safe closed IA terms. Then:*

$$\begin{aligned} M \sqsubseteq_s N &\iff [\![M]\!]_{\mathcal{I}} \lesssim_{\mathcal{I}_b} [\![N]\!]_{\mathcal{I}} \\ &\iff \mathcal{O}([\![M]\!]_{\mathcal{I}}) \lesssim_{\mathcal{I}_b} \mathcal{O}([\![N]\!]_{\mathcal{I}}) \ . \end{aligned}$$

where the preorder $\sqsubseteq_s$ is defined similarly as for safe PCF.

Proof. This result follows from the definability result as in the case of safe PCF. □

6.6 Algorithmic game semantics

The game model of safe IA is greatly simplified by the fact that justification pointers are unnecessary. By the Characterization Theorem (Sec. 2.3.7), observational equivalence of IA terms is characterized by equality of the set of complete plays. Thus for safe terms, observational equivalence is characterized by equality of the set of underlying move-sequences without justification pointers. This simplification suggests applications in algorithmic game semantics.

We show here that up to order 3, IA is a conservative extension of safe IA in the sense that the observational equivalence relations $\cong_s$ and $\cong$ coincide. Therefore, all the upper-bounds on the complexity of observational equivalence that are known for the order-3 fragments of IA also hold for safe IA. We then show that the Characterization Theorem also holds for observational equivalence of safe IA with respect to safe contexts: two terms are $\cong_s$-equivalent if the sets of complete plays of their denotation are the same. Consequently, we can show that up to order 3, the complexity lower-bounds that are already known for IA also hold in safe IA.

Observational equivalence

Proposition 6.86.

(i) Up to order 3, it is conservative, with respect to observational equivalence, to add unsafe context to safe ones. Formally for every closed IA terms M, N we have:

$$M \sqsubseteq_s N \iff M \sqsubseteq N \ .$$

(ii) Adding unsafe context is not conservative at order 4 for Idealized Algol.

Proof. (i) Let A be an order-3 type and M, N be two IA terms of type A.

$$\begin{aligned} M \sqsubseteq N &\iff [\![M]\!] \lesssim [\![N]\!] && \text{by full abstraction of IA.} \\ &\iff [\![M]\!] \lesssim_{\mathcal{I}} [\![N]\!] && \text{Corollary 6.68} \\ &\iff M \sqsubseteq_s N && \text{by full abstraction of IA w.r.t. safe contexts.} \end{aligned}$$

(ii) The idea is to start from some term E and construct a term D that behaves like E except that it has a "hidden" behaviour which can only by triggered when the Opponent plays in some particular way that is not incrementally justified. Take the following order-4 IA terms:

$$\begin{aligned} &E \equiv \lambda\varphi^{(2,2,0)}.\varphi(\lambda u_1^o.u_1\,\mathtt{skip})(\lambda u_2^o.u_2\,\mathtt{skip}) : ((2,2,0),0) \\ &D \equiv \lambda\varphi^{(2,2,0)}.\mathtt{new}\ LAST := 0\ \mathtt{in} \\ &\qquad \varphi\ (\lambda u_1^o.\mathtt{new}\ PREV :=!LAST\ \mathtt{in}\ LAST := 1; u_1([!LAST = 1]); LAST := PREV) \\ &\qquad\ (\lambda u_2^o.\mathtt{new}\ PREV :=!LAST\ \mathtt{in}\ LAST := 2; u_2([!LAST = 2]); LAST := PREV) \\ &\quad : ((2,2,0),0) \end{aligned}$$

where we use the type abbreviations 0 for `com` and $k+1 = k \to \texttt{com}$ for $k \geq 0$, and for every term $T : \texttt{exp}$, the assertion operator $[T]$ is syntactic sugar for $\texttt{cond}\ T\ \Omega\ \texttt{skip}$ (*i.e.*, the term that does nothing if T evaluates to a positive number and goes into an infinite loop otherwise).

The two terms M and N are not observationally equivalent in PCF because the unsafe context

$$C[-] = -(\lambda w_1^2 w_2^2 . w_1(\lambda x^o . w_2(\lambda y^o . \underline{x})))$$

can separate them: we have $C[D] \not\Downarrow$ and $C[E] \Downarrow$. In safe PCF, however, these two terms are observationally equivalent: Let $C[-]$ be a safe context. We claim that when evaluating $C[D]$, the variable $LAST$ always contains the index of the last called φ's parameter and therefore the assertion tests in D always succeed. This can be shown by induction on the length of the interaction play between $[\![C[-]]\!]$ and $[\![D]\!]$. We give here an informal argument. Assume that the context makes a single call to D. (The argument can be easily adapted to the general case.) During the evaluation, whenever a parameter of φ is called, D first sets the variable $LAST$ to the parameter index i and then calls the Opponent's parameter u_i. At that point, O can either make another call to one of φ's parameter, or it can call the parameter of some previous call to some u_j for $j \in \{1,2\}$. Suppose it does the latter, because it is playing incrementally (since the context is safe) such u_j is necessarily the u_i that was last called by P. The next step executed by P is then the assertion test which necessarily succeeds because $LAST$ was just set to i. When the call to u_i returns, P restores $LAST$ to the value it originally contained when φ's parameter was called, thus ensuring that it holds the index of the φ's parameter that was last called by the context.

Similarly, whenever a call to one of φ's parameter returns, the Opponent can call the parameter of the *last* (because O plays incrementally) called u_j. Since $LAST$ contains the last called φ's parameter's index, this again ensures that the assertion test succeeds. □

Characterization Theorem

We now show that a version of the Characterization Theorem (Sec. 2.3.7) also holds for safe IA:

Theorem 6.87 (Characterization Theorem in $\mathcal{I}$)**.** *Let σ and τ be two closed P-i.j. strategies on a simple game A in $\mathcal{I}$. Then*

$$\sigma \lesssim_{\mathcal{I}} \tau \qquad \iff \qquad comp(\mathcal{O}(\sigma)) \subseteq comp(\mathcal{O}(\tau)) \ .$$

Proof. By Theorem 6.85, $\sigma \lesssim_{\mathcal{I}} \tau$ iff $\mathcal{O}(\sigma) \lesssim_{\mathcal{I}} \mathcal{O}(\tau)$. The rest of the proof then follows the same argument used to prove the original Characterization Theorem in the category $\mathcal{C}_b$ [AM97, Theorem 25], with one subtlety: in the first part of the proof, the fact that $\mathcal{O}(\sigma)$ is O-i.j. guarantees that the strategy $\alpha : A \to \Sigma$ which "follows the script of s" is P-incrementally justified. □

Consequently, observational equivalence of safe IA terms with respect to safe IA contexts is characterized by equality of the sets of complete plays.

Classification

Upper bounds By Proposition 6.86, all the known upper-bound for IA are also valid for safe IA up to order 3: safe IA_2 + while is decidable in PSPACE [GM00], IA_3 + while is decidable in EXPTIME for terms in β-nf [MW05], and $\mathsf{IA}_3 + \mathsf{Y}_0$ is decidable with a complexity that is at most doubly exponentially larger than that of the DPDA equivalence problem [MOW05].

Lower bounds

Theorem 6.88 (Ong [Ong02])**.** *Observational equivalence of $IA_2 + \mathsf{Y}_1$ is undecidable.*

The proof of this theorem proceeds by reduction of the QUEUE-HALTING problem to the observational equivalence of two $\mathrm{IA}_2 + \mathsf{Y}_1$ programs: Given a QUEUE program P, a term $\vdash M_P : \mathtt{com}$ of $\mathrm{IA}_2 + \mathsf{Y}_1$ is defined such that M_P simulates P in the sense that P terminates if and only if M_P is equivalent to $\mathtt{skip}$. It turns out that the encoding term M_P [Ong02] is safe therefore:

Corollary 6.89. *Observational equivalence of safe $IA_2 + \mathsf{Y}_1$ is undecidable.*

For IA_3 + while, it was shown that the Containment Problem for DPDA can be reduced to observational approximation in $\mathrm{IA}_1 + \mathsf{Y}_0$ [MOW05, Proposition 1]. Therefore observational approximation is undecidable for $\mathrm{IA}_1 + \mathsf{Y}_0$ terms, and observational equivalence is at least as hard as DPDA Equivalence.

Corollary 6.90. *For safe $IA_2 + \mathsf{Y}_0$, observational approximation is undecidable and observational equivalence is at least as hard as DPDA Equivalence.*

Proof. The original encoding [MOW05] is not safe because it contains an occurrence of a variable $x : \mathtt{exp}$ occurring in the body of a μ-abstraction μz with $\operatorname{ord} z = \operatorname{ord} x$. An equivalent safe encoding can be obtained by replacing the free variable $x : \mathtt{exp}$ by a variable of type $\mathtt{exp} \rightarrow \mathtt{exp}$, thus giving an encoding in safe $\mathrm{IA}_2 + \mathsf{Y}_0$.

Let $\mathcal{B}$ be a DPDA over an alphabet Σ. We write $N(\mathcal{B})$ to denote the language accepted by $\mathcal{B}$. We identify values of type $\mathtt{exp}$ with $\Sigma \cup \{0\}$ and we consider the game $G = (\mathtt{exp}^2 \rightarrow \mathtt{exp}^1) \rightarrow \mathtt{com}$ whose set of moves is given by $M_G = \{q^1, q^2\} \cup \Sigma \cup \{\mathtt{run}, \mathtt{done}\}$. Following [MOW05], for every language $L \subseteq \Sigma^*$, we define $\widehat{L} \subseteq M_G^*$ as $\widehat{L} = \{\mathtt{run}\ q^1 q^2 0\, x_1 \cdots q^1 q^2 0\, x_n\ \mathtt{done}\ | x_1 \ldots x_n \in L\}$. We have $\widehat{L}_1 = \widehat{L}_2$ iff $L_1 = L_2$.

Claim: There exists a safe term $z : \mathtt{exp}^2 \rightarrow \mathtt{exp}^1 \vdash_{\mathsf{s}} Q_{\mathcal{B}} : \mathtt{com}$ such that the set of underlying sequence of moves of the complete plays of $[\![z : \mathtt{exp}^2 \rightarrow \mathtt{exp}^1 \vdash_{\mathsf{s}} Q_{\mathcal{B}} : \mathtt{com}]\!]$ is equal to $\widehat{N}(\mathcal{B})$. This term $Q_{\mathcal{B}}$ is obtained from the term $M_{\mathcal{B}}$ (from the original encoding) by replacing the free variable $x : \mathtt{exp}$ in $M_{\mathcal{B}}$ by a variable z of type $\mathtt{exp} \rightarrow \mathtt{exp}$ and by replacing the subterm "$CH := x$" by "$CH := z\, 0$". We can then conclude as in the proof for $\mathrm{IA}_1 + \mathsf{Y}_0$ [MOW05]. ☐

For IA_3 + while, Murawski et al. showed that observational equivalence is EXPTIME-hard by a reduction from the equivalence problem of nondeterministic automata on binary trees [MW05, Corollary 2]. The encoding used in the paper is unsafe but it can be easily changed into an equivalent safe term of the same order using the same trick as in the previous proof. (The variable $y : \mathtt{exp}$ is replaced by $y : \mathtt{exp} \rightarrow \mathtt{exp}$ and "$Z := y$" is replaced by "$Z := y\, 0$"). Hence:

Proposition 6.91. *Observational equivalence in safe IA_3 + while is EXPTIME-hard.*

At order 4, since adding unsafe context is not conservative (Prop. 6.86) we need to distinguish two problems: deciding $\cong$-equivalence and deciding $\cong_s$-equivalence (*i.e.*, observational equivalence defined with respect to safe contexts only).

Murawski showed that from order 4 onwards $\cong$-observational equivalence is undecidable [Mur03, Mur05a] . He considered Γ-machines, a Turing complete class of devices, and showed that for every such machine, there is an IA_4-term M such that the machine accepts the empty string if and only if the set of complete plays of $[\![M]\!]$ is not empty. This shows that $\cong$-observational equivalence is undecidable. It turns out that Murawski's encoding is safe, therefore:

Corollary 6.92. *$\cong$-observational equivalence for safe IA_4 is undecidable.*

The fact that contexts are not restricted to be safe is crucial in this simulation. The *ADD* operation of Γ-machines is for instance simulated using plays that are not O-i.j.[2] Thus the same argument can be used to show undecidability of $\cong_s$-observational equivalence. We make the following conjecture:

Conjecture 6.93. *$\cong_s$-observational equivalence for safe IA_4 is decidable.*

[2]These are the plays ending with the move r_4 in the original paper [Mur05a].

The idea is that by the Characterization Theorem for safe IA (Theorem 6.87), two terms are equivalent iff the sets of complete O-incrementally justified plays of their denotation are equal. But for such plays, all the pointers can be uniquely recovered from the underlying sequence of moves. Therefore observational equivalence is characterized by equality of the sequences of moves underlying the sequence of complete O-i.j. plays. I suspect that at order 4, such sequences can be represented by a DPDA. This would imply the above conjecture.

All the previous results are recapitulated in Table 6.6.

<table>
<tr><th></th><th></th><th colspan="5">Finitary fragments</th></tr>
<tr><th>L</th><th>Obs. eq.</th><th>order 2
+ while</th><th>order 2
+ Y_1</th><th>order 3
+ while</th><th>order 3
+Y_0</th><th>order 4</th></tr>
<tr><td rowspan="2">IA</td><td>$\cong$</td><td rowspan="4">PSPACE$^{(1)}$
$\preccurlyeq$ DFA</td><td rowspan="4">U$^{(2)}$</td><td rowspan="4">EXPTIME-hard$^{(3)}$
EXPTIME-complete for β-nf
$\preccurlyeq$ VPA</td><td rowspan="4">D$^{(4)}$
$\preccurlyeq_{exp}$ DPDA
$\succcurlyeq$ DPDA</td><td>U$^{(5)}$</td></tr>
<tr><td>$\cong_s$</td><td>?$^{(6)}$</td></tr>
<tr><td rowspan="2">Safe IA</td><td>$\cong$</td><td>U</td></tr>
<tr><td>$\cong_s$</td><td>?</td></tr>
</table>

U = Undecidable $\qquad$ $\preccurlyeq P$ = "reducible to problem P"
D = Decidable with unknown complexity $\qquad$ $\succcurlyeq P$ = "at least as hard as problem P"

(1) [GM00] (2) [Ong02] (3) [MW05] (4) [MOW05] (5) [Mur05a] (6) The Characterization Theorem does not hold in that case.

Table 6.2: Complexity of observational equivalence for finitary fragments of safe IA.

Expressivity of safe IA

Murawski introduced a notion of representability of languages by IA terms [Mur03, Mur05a] where a language is represented by (some erasure homomorphism of) the set of complete plays of the term. He showed that the class of languages representable by second-order terms is precisely the regular languages; for third-order terms it is the class of context-free languages; and for terms of order 4 and above, it is the full class of recursively enumerable languages. These results are recapitulated in Table 6.6.

What are the representable languages in the safe fragments of IA? It turns out that up to order 3, the safety constraint does not alter expressivity:

Proposition 6.94. *For $0 \leq k \leq 3$, safe IA$_k$ and IA$_k$ are equi-expressive (in terms of Murawski-representable language).*

Proof. Unsafety only appears at order 3 therefore the same languages are representable in IA$_i$ and safe IA$_i$ for $i < 3$. The order-3 term used by Murawski's encoding [Mur03, Mur05a] to

Fragment	Representable languages	Machine equivalent
IA$_0$	Singleton sets + Empty set	–
IA$_1$	Finite languages with the prefix property	–
IA$_2$	Regular languages	Finite state automata
IA$_3$	Context free languages	Pushdown automata
IA$_4$	Recursively enumerable languages	Turing machines

Table 6.3: Murawski representability.

represent context-free languages is unsafe, but it can made be easily turned into a safe term by replacing the variable $c : \texttt{exp}$ by a variable of type $(\texttt{com} \rightarrow \texttt{com}) \rightarrow \texttt{exp}$ and changing the code "$INPUT := c$" into "$INPUT := c\,(\lambda z.z)$". $\square$

It is not known which languages are expressible in higher-order fragments of safe IA. Recall that regular languages are the languages definable by 0-DPDAs, and context-free languages are those definable by DPDAs, so a possible conjecture is: "Murawski-representable in safe IA_n for $n \geq 2$ are the $(n-2)$-DPDA definable word languages". It is not clear, however, how to interpret the higher-order "push" DPDA instructions in terms of game-semantic moves. If such result were to be proven then the question of decidability of higher-order DPDA would become relevant to the observational equivalence problem: the undecidability of the former would imply that of the latter.

Chapter 7

Conclusion

7.1 Summary of contribution

Safety is a syntactic constraint for higher-order grammars. A grammar is *safe* if the right-hand side of each rule is such that no subterm occurring in operand position contains parameters of order smaller than the order of the subterm. Motivated by the appealing algorithmic properties of safety, we derived a new typing system, the *safe lambda calculus*, by imposing this syntactic constraint on the simply-typed lambda calculus. The salient property of this calculus is that it is not necessary to rename variables when performing substitution. Thus in some sense, safe terms are "easier" to compute than unsafe ones. Computation in our calculus is standardly done via the concept of β-reduction. Safety is not preserved by beta-reduction in general, but it is preserved when sufficiently many consecutive redexes are contracted simultaneously. This is formalized by the notion of *safe beta-reduction*: If a safe term contains a β-redex then this redex can always be "enlarged" into a group of consecutive beta-redexes, called a safe redex, such that contracting all of them produces a safe term. The notion of normal form thus remains unchanged. Further, safety is an extensional property: a term is safe if and only if its eta-long normal form is.

The typing system of the safe lambda calculus has desirable properties: the type-checking (Can a given type be assigned to a given term?) and typability (Given a term, is there a type that can be assigned to it?) problems are both decidable. On the other-hand, we only know that the type-inhabitation problem (Given a type, is there a safe term of that type?) is at least semi-decidable (there is an algorithm that tells if a type is inhabited by a safe term in a finite amount of time if it is the case, but may not terminate otherwise).

The loss of expressivity incurred by safety can be characterized in terms of expressible numeric functions: they are precisely the multivariate polynomials; the conditional operator, which is definable in the lambda calculus, is not expressible by any safe term. In terms of representable word functions, these are given by the set containing the projections, constant functions, concatenation and substitution and closed by composition.

We then looked at the complexity of the calculus by considering the beta-equivalence problem: we hinted that it probably lies in the complexity class ELEMENTARY by showing how both Statman and Mairson's encoding of finite type theory in the simply-typed lambda calculus fail in the safe fragment. We further showed that the problem is PSPACE-hard.

An independent contribution of this thesis is the development of a new presentation of game semantics based on the theory of traversals [Ong06a]. Essentially, traversals implement a version of β-reduction in which beta-redexes are computed locally as opposed to a global approach based on substitution. The soundness of the traversal theory as a model of computation is ensured by the correspondence with game semantics: traversals are just uncovering of plays from game semantics.

Seeking a semantic explanation of the safety constraint, we focused on the analysis of the game semantics of safe terms. We first tackle the problem by a syntactic argument based on our concrete presentation of game semantics. A notable property of safe terms is that its variables are incrementally-bound: the binder of a variable node x in the computation tree is precisely the last lambda node in the path from x to the root with order strictly greater than $\operatorname{ord} x$. By the Correspondence Theorem, this implies that the strategy denotation of a safe term is *P-incrementally justified*. In such strategy, a P-question's justifier is given by the last O-move in the P-view with greater order.

In the last chapter we finally investigated the categorical model of the safe lambda calculus. We proposed the notion of Incremental Closed Category (ICC) that soundly interprets the safe lambda calculus in the same way Cartesian Closed Categories model the simply-typed lambda calculus. We then exhibited such an ICC by constructing a game model of P-incrementally justified strategies. We showed in particular that P-incremental justified strategies compose. This gives a complete account of the game semantics of the safe lambda calculus.

To conclude, we looked at safety from the point of view of *algorithmic game semantics*. We considered the problem of observational equivalence of IA term with respect to *safe* contexts. By suitably constraining O-moves by the dual notion of *O-incremental justification*, we obtain a model of safe PCF and safe IA that is fully abstract with respect to this notion of observational equivalence. Furthermore, the model is effectively presentable: two safe terms are observationally equivalent (with respect to safe contexts) if and only if their denotations have the same set of *complete O-incrementally justified* plays.

Up to order 3, the addition of unsafe contexts to safe ones is conservative with respect to observational equivalence. Furthermore, all the complexity results—lower and upper bounds—known about observational equivalence of the (unrestricted) lower-order fragments of IA also hold in the safe sub-fragments. At order-4, however, the notion of observational equivalence with respect to unrestricted contexts differs from the one defined with respect to safe contexts only. Concerning the latter, we conjecture that the restriction of the problem to *safe* terms (*i.e.*, safe observational equivalence of safe IA_4 terms) is reducible to the DPDA-equivalence problem (which is decidable).

7.2 Further works

The nature of the safe lambda calculus is still not entirely understood. Some questions remain about the safe lambda calculus pertaining for instance to its computational power, the complexity classes that it characterizes and its interpretation under the Curry-Howard isomorphism. We now propose possible directions for further works and highlight some open questions.

Type theory

One of the most pressing open problems concerns the complexity of the safe lambda calculus. We have shown that the beta-equivalence problem is PSPACE-hard, but this lower-bound may be very coarse. Further investigations need to be done to determine an upper-bound.

Another open problem is the question of decidability of type inhabitation. At the moment we already know that it is semi-decidable: there is an algorithm that, given a simple type, can exhibit a safe inhabitant if it exists but may not terminate otherwise.

Extensions

We have defined a notion of safety for simply-typed terms (and also for untyped terms by means of a Curry-like version of the typing system). Can this be generalized to more complicated typing systems such as the second-order lambda calculus?

Logic

What kind of reasoning principles does the safe lambda calculus support via the Curry-Howard Isomorphism? How expressive is the safe fragment of intuitionistic implication logic? Is the logic decidable?—or equivalently is type inhabitation decidable in the safe lambda calculus?

Computational complexity

Can the safe lambda calculus help to characterize complexity classes? There are already many such results in the unrestricted case: Leivant and Marion [LM93] considered for instance an "impure" variation of the simply-typed lambda calculus extended with constructors, destructors and conditionals, and obtain several characterization of the polytime-computable numeric functions in that language.

Hillebrand, Kanellakis and Mairson [HKM96] considered the problem from a database point of view. Instead of encoding numeric functions, they looked at the database queries that are encodable in the simply-typed lambda calculus and gave a precise characterization of PTIME: The polynomial time queries are those expressible in the 4^{th} order fragment of the simply-typed lambda calculus. This result was later extended to give characterizations of the standard complexity classes PSPACE, k-EXPTIME, k-EXPSPACE ($k \geq 1$) and ELEMENTARY at higher-orders [HK96].

More research needs to be done to see if similar characterizations can be obtained in the safe lambda calculus.

Expressibility

Functions over free algebras

What are the function over free-algebras *definable in the safe simply-typed lambda calculus?*

There is an isomorphism between binary trees and closed simply-typed terms of type $\tau = (o \rightarrow o \rightarrow o) \rightarrow o \rightarrow o$. Thus a closed term of type $\tau \rightarrow \tau \rightarrow \ldots \rightarrow \tau$ represents an n-ary function over trees. Zaionc [Zai88] and Leivant [Lei93] gave a characterization of the set of tree functions representable in the simply-typed lambda calculus: it is precisely the minimal set containing constant functions, projections and closed under composition and limited primitive recursion. Zaionc showed that the same characterization holds for the general case of functions expressed over free algebras [Zai91]: they are given by the minimal set containing constant functions, projections and closed under composition and limited primitive recursion. This result subsumes Schwichtenberg's result on definable numeric functions as well as Zaionc's own results on definable word and tree functions.

All these basic operations are safe except limited primitive recursion. This suggests that one needs to restrict further the primitive recursion in order to obtain a characterization of free-algebra functions representable in the safe lambda calculus. Such result would generalize our expressivity result for numeric and word functions from Sec. 3.3.

Murawski-expressibility

Murawski introduced a notion of language expressibility by game semantics [Mur03, Mur05a]. He showed that the 4^{th} order finitary fragment of IA is expressive enough to give the full class of recursively enumerable languages. Does the safe fragment have the same expressive power? Another line of research would be to investigate whether the class of word languages recognizable by higher-order pushdown automata can be characterized in Murawski's sense by some higher-order fragment of safe IA.

Trees and word languages

The impact of safety on the expressivity of higher-order recursion schemes was studied in de Miranda's thesis [dM06]. At order 2 and for word languages, safety is not a genuine constraint if we allow non-determinism [AdMO05b]; de Miranda and Urzyczyn conjectured that for *deterministic* higher-order grammars, safety is a proper restriction. Urzyczyn even proposed an unsafe deterministic higher-order recursion scheme generating a word language that he conjectured to be inherently unsafe (*i.e.*, that cannot be generated by any deterministic safe grammar). At the time of this writing, though, this remains a conjecture. The traversal theory seems to be a promising tool to investigate the problem.

Game semantics

Is the game model of safe PCF universal? (*i.e.*, is every recursive incremental strategy denoted by some safe PCF term?) Is there a category of O-incrementally justified strategies?

Compilation of safe recursion schemes to pushdown automata

We have mentioned before the equi-expressivity result about safe homogeneously-typed higher-order recursion schemes and higher-order pushdown automata: these two devices generate the same class of infinite trees. Hague et al. generalized this result to unrestricted recursion scheme; one direction relies on the traversal theory: an order-n recursion scheme can be compiled into an equivalent order-n *collapsible* pushdown automaton which proceeds by computing the set of traversals of the recursion scheme's computation graph [HMOS08]. We conjecture that when the safety constraint is imposed, this encoding can be specialized into a higher-order pushdown automaton (without the collapse operation). Such result would give an alternative proof of Knapik et al.'s equi-expressivity result [KNU02].

Algorithmic game semantics

Is observational equivalence for safe *IA_4 decidable?* We have seen that up to order 3, the problem of observational equivalence has the same complexity in the safe finitary fragments as in the unrestricted finitary fragments. At order 4 the picture remains unclear. Murawski [Mur03, Mur05a] showed the undecidability of program equivalence in IA_i for $i \geq 4$ by encoding Turing machine computations using finitary IA_4 terms. Because his encoding relies on unsafe terms, the argument cannot be transposed to the safe fragment of IA. The question of whether observational equivalence of safe IA_4 is decidable thus remains open.

PUR languages

In this thesis, we have shown that the safety constraint produces languages whose game semantics enjoy the property that some justification pointers are uniquely recoverable from the underlying sequence of moves. Safe IA_3 is an example of language in which *all* pointers are recoverable. We name this class *PUR* for "*Pointer Uniquely Recoverable*". Finitary IA_2 (finite base types and no recursion) is the paradigmatic example of a PUR-language (The fact that it is a sublanguage of Safe IA_3 is another proof of this fact). But safe fragments are clearly not the only PUR-languages: singleton languages (*i.e.*, containing only one term) are trivial examples of PUR languages. Also the language consisting of all IA_3 terms whose beta-reduction is safe is also a PUR language.

A more interesting example is *Serially Re-entrant Idealized Algol* [Abr01], a version of IA where multiple uses of arguments are allowed only if they do not "overlap in time". In the game semantics denotation of a SRIA term there is at most one pending occurrence of a question at any time. Each move has therefore a unique justifier and consequently justification pointers may be ignored. Safe IA is not a sublanguage of SRIA. One reason for this is that none of the two

Kierstead terms $\lambda f.f(\lambda x.f(\lambda y.y))$ and $\lambda f.f(\lambda x.f(\lambda y.x))$ are Serially Re-entrant whereas the first one is safe. Conversely, SRIA is not a sublanguage of safe IA since the term $\lambda fg.f(\lambda x.g(\lambda y.x))$ where $f, g : ((o, o), o)$ belongs to SRIA but not to safe IA.

Another way to generate PUR-languages may consist in constraining types. Joly introduced a notion of "complexity" for lambda-terms and proved that there is a constant bounding the complexity of every closed normal lambda-term of a given type T if and only if T can be generated from a finite set of combinators. Consequently, the only inhabited finitely generated types are the types of order ≤ 2 and the types $(A_1, A_2, \ldots, A_n, o)$ such that for all $i = 1..n$: $A_i = o$, $A_i = o \rightarrow o$ or $A_i = (o^k \rightarrow o) \rightarrow o$ [Jol01]. We already know that imposing the first type restriction to Finitary IA leads to a PUR language. Does the second restriction also give rise to a PUR language?

With a view to algorithmic game semantics and its applications, the PUR class is of particular interest. Indeed, PUR-languages are good candidates of languages with decidable observational equivalence. This is because the simplification of the game-semantic model resulting from the nonnecessity of pointers makes the observational equivalence problem more manageable: in IA, for instance one just need to compare the set of complete plays underlying the denotation of a term, forgetting the justification pointers altogether. For lower-order fragments, a machine characterization of this set is sometimes possible (*e.g.*, finite-state automaton at order-2, and deterministic pushdown automata for the order-3 fragment with Y_0 recursion), subsequently leading to decidability and complexity results for the observational equivalence problem.

Bibliography

[Abr01] S. Abramsky – Semantics via game theory. In *Marktoberdorf International Summer School*, 2001, Lecture slides.

[ADLR94] A. Asperti, V. Danos, C. Laneve and L. Regnier – Paths in the lambda-calculus. In *LICS*, IEEE Computer Society, 1994, p. 426–436.

[AdMO04] K. Aehlig, J. G. de Miranda and C.-H. L. Ong – Safety is not a restriction at level 2 for string languages. Tech. report, University of Oxford, 2004.

[AdMO05a] — , The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In *TLCA* (P. Urzyczyn, ed.), Lecture Notes in Computer Science, vol. 3461, Springer, 2005, p. 39–54.

[AdMO05b] — , Safety is not a restriction at level 2 for string languages. In *FoSSaCS* [Sas05], p. 490–504.

[AGOM03] S. Abramsky, D. R. Ghica, C.-H. L. Ong and A. Murawski – Algorithmic Game Semantics and Component-Based Verification. In *Proceedings of SAVBCS 2003: Specification and Verification of Component-Based Systems, Workshop at ESEC/FASE 2003*, 2003, published as Technical Report 03-11, Department of Computer Science, Iowa State University, p. 66–74.

[AHM98] S. Abramsky, K. Honda and G. McCusker – A fully abstract game semantics for general references. In *Logic in Computer Science, 1998. Proceedings. Thirteenth Annual IEEE Symposium on*, 1998, p. 334–344.

[Aho68] A. V. Aho – Indexed grammars – an extension of context-free grammars. *J. ACM* **15** (1968), no. 4, p. 647–671.

[AJ92] S. Abramsky and R. Jagadeesan – Games and full completeness for multiplicative linear logic. In *Foundations of Software Technology and Theoretical Computer Science (FST-TCS'92)* (New Delhi, India) (R. Shyamasundar, ed.), 1992, p. 291–301.

[AJ05] — , A game semantics for generic polymorphism. *Ann. Pure Appl. Logic* **133** (2005), no. 1-3, p. 3–37.

[AM97] S. Abramsky and G. McCusker – Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In *Algol-like languages* (P. W. O'Hearn and R. D. Tennent, ed.), Birkhaüser, 1997.

[AM98a] — , Call-by-value games. In *Computer Science Logic: 11th International Workshop Proceedings* (M. Nielsen and W. Thomas, ed.), Springer-Verlag, 1998.

[AM98b] — , Game semantics. In *Logic and Computation: Proceedings of the 1997 Marktoberdorf Summer School* (H. Schwichtenberg and U. Berger, ed.), Springer-Verlag, 1998, Lecture notes, p. 1–56.

[AM99] — , Full abstraction for Idealized Algol with passive expressions. *Theoretical Computer Science* **227** (1999), no. 1–2, p. 3–42.

[AMJ94] S. Abramsky, P. Malacaria and R. Jagadeesan – Full abstraction for PCF. In *Theoretical Aspects of Computer Software*, 1994, p. 1–15.

[Asp] A. Asperti – P = NP, up to sharing.

[Bar84] H. P. Barendregt – *The Lambda Calculus – Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics, vol. 103, North-Holland, 1984.

[Bar92] H. Barendregt – Lambda calculi with types. In Handbook of Logic in Computer Science (S. Abramsky, D. M. Gabbay and T. Maibaum, ed.), vol. 2, Clarendon Press, 1992, p. 117–309.

[BC82] G. Berry and P.-L. Curien – Sequential algorithms on concrete data structures. *Theoretical Computure Science* **20** (1982), p. 265–321.

[Ber78] G. Berry – Stable models of typed lambda-calculi. In *Proceedings of the Fifth Colloquium on Automata, Languages and Programming* (London, UK), Springer-Verlag, 1978, p. 72–89.

[Ber79] — , Modèles complément adéquats et stable des lambda-calculs typés. Phd thesis, Université Paris VII, 1979.

[Bla92] A. Blass – A game semantics for linear logic. *Annals of Pure and Applied Logic* **56** (1992), no. 1-3, p. 183–220.

[Blu08] W. Blum – A tool for constructing structures generated by higher-order recursion schemes and collapsible pushdown automata. http://william.famille-blum.org/research/tools.html, 2008.

[BO07] W. Blum and C.-H. L. Ong – The safe lambda calculus. In *TLCA* (S. R. D. Rocca, ed.), Lecture Notes in Computer Science, vol. 4583, Springer, 2007, p. 39–53.

[Cau02] D. Caucal – On infinite terms having a decidable monadic theory. *Lecture Notes in Computer Science* **2420** (2002), p. 165–176.

[CH06] F. Cardone and J. R. Hindley – History of lambda-calculus and combinatory logic. *Handbook of the History of Logic* **5** (2006).

[CM64] J. Cocke and M. Minsky – Universality of tag systems with p = 2. *J. ACM* **11** (1964), no. 1, p. 15–20.

[Cro93] R. Crole – *Categories for types*. Cambridge Mathematical Textbooks, Cambridge University Press, 1993.

[Dam82] W. Damm – The IO- and OI-hierarchy. *TCS* **20** (1982), p. 95–207.

[DG86] W. Damm and A. Goerdt – An automata-theoretical characterization of the OI-hierarchy. *Information and Control* **71** (1986), no. 1-2, p. 1–32.

[DGL05] A. Dimovski, D. R. Ghica and R. Lazic – Data-abstraction refinement: A game semantic approach. In *SAS* (C. Hankin and I. Siveroni, ed.), Lecture Notes in Computer Science, vol. 3672, Springer, 2005, p. 102–117.

[DHR96] V. Danos, H. Herbelin and L. Regnier – Game semantics and abstract machines. In *Logic in Computer Science, 1996. LICS '96. Proceedings., Eleventh Annual IEEE Symposium on*, 27-30 July 1996, p. 394–405.

[dM06] J. G. de Miranda – Structures generated by higher-order grammars and the safety constraint. D.Phil thesis, University of Oxford, 2006.

[DR93] V. Danos and L. Regnier – Local and asynchronous beta-reduction (an analysis of girard's execution formula). In *Proceedings of the Eighth Annual IEEE Symp. on Logic in Computer Science, LICS 1993* (M. Vardi, ed.), IEEE Computer Society Press, June 1993, p. 296–306.

[DR04] — , Head linear reduction. submitted for publication, 2004.

[FLO83] S. Fortune, D. Leivant and M. O'Donnell – The expressiveness of simple and second-order type structures. *J. ACM* **30** (1983), no. 1, p. 151–185.

[GM00] D. R. Ghica and G. McCusker – Reasoning about idealized algol using regular languages. In *Proceedings of 27th International Colloquium on Automata, Languages and Programming ICALP 2000*, LNCS, vol. 1853, Springer-Verlag, 2000, p. 103–116.

[GM03] D. R. Ghica and G. McCusker – The regular-language semantics of second-order Idealized Algol. *Theoretical Computer Science* **309** (2003), no. 1-3, p. 469–502.

[Gre04] W. Greenland – Game semantics for region analysis. Ph.D. thesis, University of Oxford, 2004.

[Har05] R. Harmer – Innocent game semantics. November 2005, Course notes.

[Hin97] J. R. Hindley – *Basic simple type theory*. Cambridge University Press, New York, NY, USA, 1997.

[HK96] G. G. Hillebrand and P. C. Kanellakis – On the expressive power of simply typed and let-polymorphic lambda calculi. In *LICS*, 1996, p. 253–263.

[HKM96] G. G. Hillebrand, P. C. Kanellakis and H. G. Mairson – Database query languages embedded in the typed lambda calculus. *Inf. Comput.* **127** (1996), no. 2, p. 117–144.

[HMOS08] M. Hague, A. S. Murawski, C.-H. L. Ong and O. Serre – Collapsible pushdown automata and recursive schemes. *LICS* (2008), p. 452–461.

[HO93] J. M. E. Hyland and C.-H. L. Ong – Fair games and full completeness for Multiplicative Linear Logic without the MIX-rule. preprint, 1993.

[HO00] — , On full abstraction for PCF: I, II, and III. *Information and Computation* **163** (2000), no. 2, p. 285–408.

[Hoa83] C. A. R. Hoare – Communicating sequential processes. *Commun. ACM* **26** (1983), no. 1, p. 100–106.

[HS86] J. R. Hindley and J. P. Seldin – *Introduction to combinators and lambda-calculus*. Cambridge University Press, 1986.

[Hue75] G. P. Huet – A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.* **1** (1975), no. 1, p. 27–57.

[Hue76] — , Résolution d'équations dans des langages d'ordre 1,2,...,ω. Thèse de doctorat es sciences mathématiques, Université Paris VII, Septembre 1976.

[HY99] K. Honda and N. Yoshida – Game-theoretic analysis of call-by-value computation. *Theoretical Computer Science* **221** (1999), no. 1–2, p. 393–456.

[Jol01] T. Joly – The finitely generated types of the lambda-calculus. In *TLCA*, 2001, p. 240–252.

[Joy77] A. Joyal – Remarques sur la théorie des jeux a deux personnes. *Gazette des Sciences Mathéematiques du Quebec* **1** (1977), p. 4.

[JP76] D. C. Jensen and T. Pietrzykowski – Mechanizing *mega*-order type theory through unification. *Theor. Comput. Sci.* **3** (1976), no. 2, p. 123–171.

[Knu00] D. E. Knuth – *Fundamental algorithms*. Third ed., The Art of Computer Programming, vol. 1, Addison-Wesley, 2000.

[KNU02] T. Knapik, D. Niwiński and P. Urzyczyn – Higher-order pushdown trees are easy. In *FOSSACS'02*, Springer, 2002, LNCS Vol. 2303, p. 205–222.

[Kob09] N. Kobayashi – Types and higher-order recursion schemes for verification of higher-order programs. Submitted to the Symposium on Principles of Programming Languages, 2009.

[Lai97] J. Laird – Full abstraction for functional languages with control. In *LICS '97: Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science* (Washington, DC, USA), IEEE Computer Society, 1997, p. 58.

[Lam86] J. Lambek – Cartesian closed categories and typed lambda-calculi. *Proc. of the thirteenth spring school of the LITP on Combinators and functional programming languages table of contents* (1986), p. 136–175.

[Lam90] J. Lamping – An algorithm for optimal lambda calculus reduction. In *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (New York, NY, USA), ACM Press, 1990, p. 16–30.

[Lei93] D. Leivant – Functions over free algebras definable in the simply typed lambda calculus. *Theor. Comput. Sci.* **121** (1993), no. 1&2, p. 309–322.

[LJBA01] C. S. Lee, N. D. Jones and A. M. Ben-Amram – The size-change principle for program termination. In *POPL*, Proceedings ACM Symposium on Principles of Programming Languages, 2001.

[LM93] D. Leivant and J.-Y. Marion – Lambda calculus characterizations of poly-time. In *TLCA* (M. Bezem and J. F. Groote, ed.), Lecture Notes in Computer Science, vol. 664, Springer, 1993, p. 274–288.

[Loa98a] R. Loader – Notes on simply typed lambda calculus. February 1998.

[Loa98b] — , Unary PCF is decidable. *Theoretical Computer Science* **206** (1998), no. 1-2, p. 317–329.

[Loa01] — , Finitary PCF is not decidable. *Theoretical Computer Science* **266** (2001), no. 1-2, p. 341–364.

[Lor61] P. Lorenzen – Ein dialogisches konstruktivitätskriterium. In *Infinitistic Methods.* (W. PWN, ed.), 1961, p. 193–200.

[Mai92] H. G. Mairson – A Simple Proof of a Theorem of Statman. *TCS* **103** (1992), no. 2, p. 387–394.

[Mas74] A. N. Maslov – The hierarchy of indexed languages of an arbitrary level. *Soviet Math. Dokl.* **15** (1974), p. 1170–1174.

[Mas76] — , Multilevel stack automata. *Problems of Information Transmission* **12** (1976), p. 38–43.

[McC96a] G. McCusker – Games and full abstraction for a functional metalanguage with recursive types. Ph.D. thesis, Imperial College, 1996.

[McC96b] — , Games and full abstraction for FPC. In *Proceedings of the Eleventh Annual IEEE Symp. on Logic in Computer Science, LICS 1996* (E. M. Clarke, ed.), IEEE Computer Society Press, July 1996, p. 174–183.

[McC03] — , On the semantics of Idealized Algol without the bad-variable constructor. In *Nineteenth Conference on the Mathematical Foundations of Programming Semantics* (ENTCS, ed.), vol. 83, Elsevier, 2003.

[Mey74] A. R. Meyer – The inherent computational complexity of theories of ordered sets. In *Proc. Int'l. Cong. of Mathematicians*, vol. 2, August 1974, p. 477–482.

[Min67] M. L. Minsky – *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[MOW05] A. S. Murawski, C.-H. L. Ong and I. Walukiewicz – Idealized algol with ground recursion, and DPDA equivalence. In *ICALP* (L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi and M. Yung, ed.), Lecture Notes in Computer Science, vol. 3580, Springer, 2005, p. 917–929.

[MP07] G. McCusker and D. J. Pym – A games model of bunched implications. In *CSL* (J. Duparc and T. A. Henzinger, ed.), Lecture Notes in Computer Science, vol. 4646, Springer, 2007, p. 573–588.

[Mur03] A. S. Murawski – On program equivalence in languages with ground-type references. In *Logic in Computer Science, 2003. Proceedings. 18th Annual IEEE Symposium on*, 22-25 June 2003, p. 108–117.

[Mur05a] — , About the undecidability of program equivalence in finitary languages with state. *ACM Trans. Comput. Logic* **6** (2005), p. 701–726.

[Mur05b] — , Games for complexity second-order call-by-name programs. *Theoretical Computer Science* **343** (2005), p. 207–236, special issue: Game Theory meets Computer Science, accepted for publication.

[MW05] A. S. Murawski and I. Walukiewicz – Third-order idealized algol with iteration is decidable. In *FoSSaCS* [Sas05], p. 202–218.

[Nic94] H. Nickau – Hereditarily sequential functionals. In *Proc. Symp. Logical Foundations of Computer Science: Logic at St. Petersburg* (A. Nerode and Y. V. Matiyasevich, ed.), Lecture Notes in Computer Science, vol. 813, Springer-Verlag, 1994, p. 253–264.

[Ong02] C.-H. L. Ong – Observational equivalence of third-order Idealized Algol is decidable. In *Proceedings of IEEE Symposium on Logic in Computer Science, 22-25 July 2002, Copenhagen Denmark*, Computer Society Press, 2002, p. 245–256.

[Ong04] — , An approach to deciding observational equivalence of algol-like languages. *Ann. Pure Appl. Logic* **130** (2004), no. 1-3, p. 125–171.

[Ong06a] — , On model-checking trees generated by higher-order recursion schemes. In *Proceedings of IEEE Symposium on Logic in Computer Science.*, Computer Society Press, 2006, Extended abstract, p. 81–90.

[Ong06b] — , On model-checking trees generated by higher-order recursion schemes (technical report). Preprint, 42 pp, 2006.

[OT] C.-H. L. Ong and N. Tzevelekos – Functional reachability. preprint.

[Plo75] G. D. Plotkin – Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science* **1** (1975), no. 2, p. 125–159.

[Plo77] — , LCF considered as a programming language. *Theor. Comput. Sci.* **5** (1977), no. 3, p. 225–255.

[Rey81] J. C. Reynolds – The essence of algol. In Algorithmic Languages (J. W. de Bakker and J. C. van Vliet, ed.), IFIP, North-Holland, Amsterdam, 1981, p. 345–372.

[Sas05] V. Sassone (ed.) – *Foundations of Software Science and Computational Structures, 8th international conference, FOSSACS 2005, held as part of the joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, proceedings.* Lecture Notes in Computer Science, vol. 3441, Springer, 2005.

[Sch76] H. Schwichtenberg – Definierbare funktionen im lambda-kalkul mit typen. *Archiv Logik Grundlagenforsch* **17** (1976), p. 113–114.

[Sch01] A. Schubert – The complexity of beta-reduction in low orders. *Proceedings TLCA 2001* (2001), p. 400–414.

[Sco69] D. S. Scott – A theory of computable function of higher type. Unpublished seminar notes, University of Oxford, 1969.

[Sco93] — , A type-theoretical alternative to iswim, cuch, owhy. *Theor. Comput. Sci.* **121** (1993), no. 1-2, p. 411–440.

[Sén01] G. Sénizergues – L(A)=L(B)? decidability results from complete formal systems. *Theor. Comput. Sci.* **251** (2001), no. 1-2, p. 1–166.

[Ser05] D. Sereni – Simply typed λ-calculus and SCT. Unpublished notes, 2005.

[Sta79a] R. Statman – Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science* **9** (1979), no. 1, p. 67–72.

[Sta79b] — , The typed lambda-calculus is not elementary recursive. *Theoretical Computer Science* **9** (1979), no. 1, p. 73–81.

[Sti02] C. Stirling – Deciding dpda equivalence is primitive recursive. In *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming* (London, UK), Springer-Verlag, 2002, p. 821–832.

[Sti06] — , A game-theoretic approach to deciding higher-order matching. In *ICALP (2)* (M. Bugliesi, B. Preneel, V. Sassone and I. Wegener, ed.), Lecture Notes in Computer Science, vol. 4052, Springer, 2006, p. 348–359.

[Tai67] W. Tait – Intensional interpretations of functionals of finite type I. *J. Symb. Log.* **32** (1967), no. 2, p. 198–212.

[Zai87] M. Zaionc – Word operation definable in the typed lambda-calculus. *Theor. Comput. Sci.* **52** (1987), p. 1–14.

[Zai88] — , On the lambda-definable tree operations. In *Algebraic Logic and Universal Algebra in Computer Science* (C. Bergman, R. D. Maddux and D. Pigozzi, ed.), Lecture Notes in Computer Science, vol. 425, Springer, 1988, p. 279–292.

[Zai91] — , Lambda-definability on free algebras. *Ann. Pure Appl. Logic* **51** (1991), no. 3, p. 279–300.

[Zai95] — , Lambda representation of operations between different term algebras. *Lecture Notes in Computer Science* (1995), p. 91–105.

Index to notations

Symbolism	**Meaning**	**Page**
$FV(M)$	Set of free variables of the term M	10
$M \equiv N$	Syntactic equality of terms (modulo α-conversion)	10
$M\{N/x\}$	Capture-permitting substitution of N for x in M	10
$M[N/x]$	Substitution of N for x in M	10
$\rightarrow_\beta$	Beta-reduction	11
$s \cdot s'$	Concatenation of the (justified) sequences s and s'	30
ϵ	The empty (justified) sequence	30
$s_{\leqslant m}$	Prefix of the (justified) sequence s ending with the occurrence m	30
$\ulcorner s \urcorner$	Proponent view of a justified sequence of move	30
$\llcorner s \lrcorner$	Opponent view of a justified sequence of move	30
$\sigma;\tau$	Linear strategy composition	34
$\sigma \fatsemi \tau$	Strategy composition	36
$[\![T]\!]$	Game denotation of a type T	37
$[\![M]\!]$	Strategy denotation of a term M	38
$\mathsf{Pref}^{\mathsf{even}} X$	Subset of even length prefixes of X	38
$C[-]$	Context with a hole denoted by $-$	39
$\rightarrow_{\beta_s}$	Safe beta-reduction	57
$\lceil M \rceil$	Eta-long normal form of the term M	58
$\tau(M)$	Computation tree of the term M	97
$\circledast$	Root of the computation tree	98
$S^{H\vdash}$	Subset of S consisting of the nodes hereditarily enabled by some node in H	100

Symbolism	Meaning	Page
$s \leqslant s'$	Prefix ordering for (justified) sequences	101
$t \upharpoonright n$	Hereditary projection of justified sequence t with respect to occurrence n	101
$t \upharpoonright\upharpoonright n$	Subterm projection of a traversal t with respect to occurrence n	111
$\mathsf{ext}(t)$	Extension of a justified sequence of nodes	115
$\mathsf{Pref}(S)$	Prefix-closure of the set S.	123
$\langle\!\langle M \rangle\!\rangle$	Revealed strategy denotation of a term M	124
$t \sqsubseteq t'$	Approximation ordering for trees	150
$[n_1, n_2]$	Path in a tree from node n_1 to node n_2	166
$s \sqsubseteq s'$	Subsequence relation for (justified) sequences	191
$s \geqslant s'$	Suffix relation for (justified) sequences	191
$\sigma \lesssim \tau$	Intrinsic preorder in the category of games $\mathcal{C}$	200
$\sigma \lesssim_{\mathcal{I}} \tau$	Intrinsic preorder in the category of games $\mathcal{I}$	200
$M \sqsubseteq\!\!\sim N$	Observational preorder	207
$M \sqsubseteq\!\!\sim_s N$	Observational preorder with respect to safe contexts	207

Index