CONTRIBUTIONS TO THE THEORY OF SYNTAX WITH BINDINGS AND TO PROCESS ALGEBRA

BY

ANDREI POPESCU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Doctoral Committee:

Research Associate Professor Elsa Gunter, Chair and Director of Research
Professor Gul Agha
Associate Professor Grigore Roşu
Professor Amy Felty, University of Ottawa

Abstract

We develop a theory of syntax with bindings, focusing on:
- methodological issues concerning the convenient representation of syntax;
- techniques for recursive definitions and inductive reasoning.
Our approach consists of a combination of FOAS (First-Order Abstract Syntax) and HOAS (Higher-Order Abstract Syntax) and tries to take advantage of the best of both worlds. The connection between FOAS and HOAS follows some general patterns and is presented as a (formally certified) statement of adequacy.

We also develop a general technique for proving bisimilarity in process algebra Our technique, presented as a formal proof system, is applicable to a wide range of process algebras. The proof system is *incremental*, in that it allows building incrementally an a priori unknown bisimulation, and *pattern-based*, in that it works on equalities of process patterns (i.e., universally quantified equations of process terms containing process variables), thus taking advantage of equational reasoning in a "circular" manner, inside coinductive proof loops.

All the work presented here has been formalized in the Isabelle theorem prover. The formalization is performed in a general setting: arbitrary many-sorted syntax with bindings and arbitrary SOS-specified process algebra in de Simone format. The usefulness of our techniques is illustrated by several formalized case studies:
- a development of call-by-name and call-by-value λ-calculus with constants, including Church-Rosser theorems, connection with de Bruijn representation, connection with other Isabelle formalizations, HOAS representation, and contituation-passing-style (CPS) transformation;
- a proof in HOAS of strong normalization for the polymorphic second-order λ-calculus (a.k.a. System F).
We also indicate the outline and some details of the formal development.

to Leili R. Marleene

Acknowledgments

I thank my adviser, Elsa Gunter. Professionally, she has inoculated me with the passion for theorem proving. More personally, but also with a strong professional component, she has provided me with one of the rare examples of people I could "safely" admire, without any reserve.

I thank my good friend, Traian Şerbănuţă, who occasionally played the role of an Alyosha Karamazov during turbid times.

I thank Grigore Roşu for his mentoring in the first half of this Ph.D., and for his continuous support and friendship. I also thank him for his heroic (but unfortunately failed) attempt to transform me into a morning person.

I thank my colleague Ayesha Yasmeen for encouraging me to develop and finish the work reported here.

I thank Professor Amy Felty for being an active member in my dissertation committee, and for her inspiring work on Higher-Order Abstract Syntax.

I thank Professor Gul Agha for the high intellectual standing of his course on concurrency – taking this course had the effect of a Copernican revolution in the way I regard the topic.

I thank Dr. Tom Gambill, at whose courses I have been TAing for a large part of this Ph.D.. He was a very reasonable and caring supervisor – this helped tremendously with my time management.

I thank my parents, back home in Romania, to whom I also apologize for taking so long to finish.

The research presented in this thesis was supported in part by the NSF Grant #0917218 TC: Small: Formalizing Operator Task Analysis.

Table of Contents

Chapter 1

Context

1.1 Introduction

Given the ever increasing complexity of modern software systems, the need for convenient theoretical frameworks for specifying, organizing, and reasoning about such systems has become drastic.

(1) A salient feature of many of these systems is the presence of *scoping* and *bindings* at the level of their syntax, reflected by *higher-order* functionals at the level of their mathematical semantics. A sound and clean conceptual setting for the scoping and binding structure typically facilitates a clean semantics and, consequently, the availability of insightful, intuitive and easy to use reasoning mechanisms. In effect, it has been widely recognized in both formal logic and programming language theory that the syntactic structure of formal systems stays in a very tight relationship with the structure of inference, and that inference itself is a process of building "generalized syntax", with binding, scoping and substitution as the main engines. Because of their highly intuitive nature, the subtleties of these engines are too often treated rather non-rigorously in mathematical textbooks, with the expectation that the reader will fill in the details. By contrast, an implementation or a formalization (of a programming language or a logic) has to give a full formal account of these concepts and consequently has to deal with a myriad of details (such as renaming variables to avoid variable capture), which tend to become overwhelming and hinder the access to general ideas or goals.

Relatively recently, quite a few logicians and computer scientists became interested in taking these "details" more seriously and organizing them on sound formal principles having in mind not only mathematical rigor, but also the possibility of their hassle-free manipulation in definitions and proofs.

First-order abstract syntax (FOAS) is an already traditional methodology for describing the syntax of logics and programming languages. Several recent approaches, notably Nominal Logic and work based on functor categories[1] are adapting/generalizing FOAS to give a

[1] To avoid loading the introduction with long lists of bracketed numbers, we do not to cite any paper in this introduction, deferring citations to the more technical parts of the text.

deeper account of the notion of binding, thus going beyond the context-freeness limitation of standard FOAS.

Another powerful methodology emerging from these efforts is the so called *Higher Order Abstract Syntax* (HOAS) approach, which tries to identify (whenever possible) object-level mechanisms with corresponding meta-level mechanisms from the underlying logic. Thus, for instance, the presumptive λ-abstraction from the object system would be represented by λ-abstraction in the meta-logic, so that object-level substitution becomes mere function application – this avoids (or, better said, integrates into the meta-level layer) a great amount of tedious details. One can notice from the above example that, in the context of the host logic being a familiar logic for the development of mathematics such as higher-order logic where bindings have a functional meaning, the HOAS approach may be regarded as an effort to *anticipate syntactically* as much as possible from the *semantics* of a language. Indeed, under HOAS, an abstract syntax tree is no longer pure syntax, but features semantic handles; thus, a term $\lambda x.E$ is now represented as an honest-to-goodness function (as its semantics would typically prescribe), able to take inputs and return a result via the substitution mechanism, which now has become function application. (Therefore, the ability to accommodate part of the intended semantics in advance into the syntax on one hand and the ability to perform hassle-free reasoning on that syntax on the other appear as two faces of the same coin.) The HOAS convenience comes with a price though: given that the object system is now integrated in the meta layer, often desired facilities such as structural inductive reasoning are no longer immediately available.

Recovering such facilities while retaining the advantages of HOAS is a subject of intensive ongoing research in the HOAS community. This is also a main theme of this dissertation, where HOAS is combined with, and based on, a FOAS representation and machinery.

(2) Another feature that becomes increasingly important these days is *concurrent behavior*, which needs to be accommodated into essentially all modern software systems. Concurrency refuses to obey many paradigms well-established for sequential functional systems, notably domain theory. The gap between the rather straightforward description of concurrent systems by Structural Operational Semantics (SOS) or other similar means and the actual intended semantics, which needs to be filled in by rather elaborate notions of process equivalence such as bisimilarity, testing equivalence or behavioral equivalence (with various flavors), is one of the difficulties in dealing with concurrent systems on a formal basis. Model-checking is a major approach to the formal analysis of concurrent systems, but has well-known limitations given by its availability mainly for finite-space systems.[2] Coalgebra theory is another framework, developed purposely to handle concurrent behavior, and provides an

[2]Techniques for attempting to construct automatically finite abstractions of infinite systems are the subject of very rapidly developing research, with significant results both for safety and for liveness – these and other approaches around model checking fall out of the scope of our work, which will be instead oriented towards theorem proving.

elegant setting for studying the various notions of behavior and behavioral equivalence of concurrent systems. The foundations of (set-theoretic) coalgebra, less demanding than those of domain theory, accommodate concurrent behavior naturally, with the *final coalgebra semantics* allowing for capturing the notion of behavior in its essential determinations, without the detour of factoring some not-yet-abstract items to a bisimilarity relation. In this dissertation, we take advantage of certain coalgebraic insights for developing a powerful conductive technique for systems in a general syntactic format and for arguing that many systems can be cast into this format.

(3) The realm of interaction between bindings and concurrency brings into the highlight new phenomena such as the (inter-related) ones of *channel passing*, *scope extrusion*, and *dynamic communication topology*. These phenomena are already mainstream in many process calculi and in some concurrent programming languages, and are also on the HOAS agenda. Frameworks combining (weak) HOAS and denotational semantics, as well as more syntax-oriented coinduction rules that take binding and substitution into account, have been developed to model this interaction. Integrating our approach to representing syntax with our coinductive techniques will be another goal of our thesis.

The work reported in this dissertation is concerned with the subject of points (1) and (2) above. We also discuss some ideas concerning point (3) along the lines of our approach to (1) and (2) and report some partial progress towards this goal, but the goal is left for future work.

Concerning point (1), we have built a framework, consisting of a mathematical theory and its formalization in Isabelle/HOL, for facilitating the specification of syntax and formal systems involving binding and substitution mechanisms. It consists of:

- A first-order theory of terms with bindings, featuring recursive and inductive principles;
- A HOAS methodology, including representation and HOAS-specific induction and recursion mechanisms; this methodology is developed within a HOAS layer on top of the first-order layer (and is called "HOAS on top of FOAS"), offering an alternative, higher-level type of access to the involved concepts.

We have worked out extensive formalized case studies based on this framework.

Concerning point (2), we have developed a general technique for proving bisimilarity in process algebra. Our technique is presented as a formal proof system, formalized in Isabelle/HOL, applicable to a large class of process algebras: those specifiable in the de Simone SOS format. The proof system is *incremental*, in that it allows building incrementally an a priori unknown bisimulation, and *pattern-based*, in that it works on equalities of process patterns (i.e., universally quantified equations of process terms containing process variables), thus taking advantage of equational reasoning in a "circular" manner, inside coinductive proof loops. We also argue that important cases not traditionally regarded as fulfilling the de

Simone format, notably process algebras under weak bisimilarity, can be cast into it, hence fall into the scope of our technique.

1.1.1 Outline of this dissertation

In the remainder of this chapter, we do the following:
- In Section 1.2, we go informally through the necessary background and review several approaches and results relevant for, or related to, our work reported here.
- In Section 1.3, we establish mathematical notation and terminology, and describe general conventions we follow throughout this dissertation.
- In Section 1.4, we discuss technical preliminaries on syntax with bindings and Horn theory.

The presentation of the contribution is divided in two parts, according to the two topics of this dissertation: syntax with bindings and process algebra. The former is dealt with in Chapters 2 and 3, and the latter in Chapter 4.

In Chapter 2, we discuss a first-order approach to syntax with bindings, i.e., one employing a first-order abstract syntax (FOAS) representation. The focus here is on one of the main difficulties encountered when working with syntax on a rigorous/formal basis: recursion definitional principles. (This difficulty is essentially due to the non-injectiveness of the binding constructs.) We propose such principles having a basis on Horn theory and initial models, and illustrate their usefulness by a long series of examples. We also discuss the formalization of a many-sorted theory that implements such principles, as well as other features reflecting the state-of-the-art from the literature, notably induction principles with freshness assumptions. We end the chapter by discussing some specific related work.

In Chapter 3, we present a higher-order approach to syntax with bindings, i.e., one employing higher-order abstract syntax (HOAS). We build a HOAS machinery *on top of FOAS*, that is to say, as a definitional extension of the FOAS framework discussed in the previous chapter. This machinery consists of a *HOAS view* on the first-order syntax and of a technique for HOAS representation of inductively defined relations, such as reduction and typing. Unlike in the previous chapter, here we only consider one example – the syntax and operational semantics of System F – which we subject thoroughly to our techniques, culminating in a fairly simple proof of the Strong Normalization property. Again, in the end we discuss formalization aspects and specific related work.

Chapter 4 contains Part II of our contribution: an incremental bisimilarity proof technique for process algebra. We describe the general theory and, as we go, we illustrate it on a working example: a mini-process calculus a la CCS. We also analyze the scope of our results, showing how to capture features like recursion and weak bisimilarity. As before, discussions of formalization aspects and related work end the chapter.

Chapter 5 draws conclusions and discusses plans for future work.

1.2 Background and some related work

Here we describe some existing work on the topic of this dissertation and position our own contribution within this body of work. The discussion is phrased in general (and sometimes vague) terms. More specific related work is discussed at the end of each of the main chapters (2, 3 and 4).

1.2.1 First-order approaches to syntax representation

A first-order view on syntax with bindings is characterized by the consideration of binding operators as first-order operations. Within first-order approaches, we can identify two main variants.

First, one where, e.g., λ-abstraction is represented directly as an operator $\mathsf{Lm} : \mathbf{var} \to \mathbf{term} \to \mathbf{term}$. The very definition of syntax (that is to say, the traditional definition) already takes this view. The fact that terms are also identified modulo α-equivalence does not impair the first order nature of binding operators, as, e.g., Lm still has the type $\mathbf{var} \to \mathbf{term} \to \mathbf{term}$ even if terms are taken to be α-equivalence classes of "raw (quasi-)terms",[3] but does impair the freeness of these operators, as they are no longer absolutely free constructors – e.g., $\mathsf{Lm}\ x\ X = \mathsf{Lm}\ y\ Y$ no longer implies $(x, X) = (y, Y)$, but only implies the existence of a fresh z such that $X[z/x] = Y[z/y]$. The fact that Lm considers variables explicitly as arguments makes variables (to a large extent) first-class citizens, just like terms. This "traditional" aspect of variables is rather counterintuitive, as bound variables should be viewed as merely *positions* in terms, whose particular individuality is irrelevant. Of course, α-equivalence is purposely targeted to "correct" this, but by itself is not able to blur all the deeper inconveniences brought by the original commitment to bind specific variables. In order to have a smooth treatment of syntax with bindings (which includes hassle-free definitional and proof principles), one needs to put more effort in organizing, and indeed in understanding this syntax. Nominal Logic [115] (also implemented in Isabelle [148]) is one approach to achieving the above, and is based on the notion of *equivariance* of the involved predicates, which essentially means that the predicates do not treat variables as fully individual entities, and are thus invariant under variable permutations. A convenient feature of the nominal setting is that producing a fresh variable (in a context that uses only finitely many variables) has really the meaning of producing *any* fresh variable, the choice being *a priori established as immaterial*. An approach that combines an α-equivalence HOL axiomatization in [58] with ideas from Nominal Logic can be found in [104]. More abstract approaches are based on functor categories [46, 67, 14], where the uniformity of

[3] Some prefer of speak of the second-order nature of terms when they are considered modulo α, in contrast with the first-order nature of "raw (quasi-)terms". Our terminology does not reflect this distinction, as it refers strictly to the representation of the binding operator.

syntax transformers w.r.t. variables is captured by natural transformations.

Second, one where abstraction is encoded by other first-order operators, as in, e.g., combinatory logic [20] or de Bruijn-style representations [36] – these approaches are useful for implementation purposes, since they avoid α-equivalence classes, but severely affect readability and in general the good control over the ideas when manipulating terms manually (e.g., in definitions and theorem proving).[4] Approaches aimed at avoiding this problem (with the expense of introducing more auxiliary operators) are mostly based on equational logic and rewriting, and include the calculus with explicit substitutions [8] and rewriting logic [83].

There are also mixed approaches, notably Gordon's [57], which use de Bruijn indexes for bindings and concrete names for free variables, and also provide machinery for converting between the two at the time of binding. Recent work [139, 118] (building on previous work [82]) advocates the so-called *locally-named approach*, which avoids de Bruijn indexes as well as α-classes altogether, by distinguishing between local variables and global variables. Some HOAS approaches (namely, the ones using a general-purpose logic) typically need a preliminary study and "taming" of FOAS syntax in the style of [57]. Moreover, regarded through the abstract filter of category theory, a first-order approach based on de Bruijn levels is essentially the same as a a so-called *weak HOAS* approach (see also Section 2.10.2).

1.2.2 The Higher-Order Abstract Syntax approach to syntax representation

By contrast to a first-order view, a higher-order view regards λ-abstraction as a second-order operation of some kind, such as $\mathsf{Lm} : (\mathbf{var} \to \mathbf{term}) \to \mathbf{term}$ or $\mathsf{Lm} : (\mathbf{term} \to \mathbf{term}) \to \mathbf{term}$ – this is known as *HOAS* (*Higher-Order Abstract Syntax*). Traditionally, HOAS is a methodology for representing formal systems (typically, logical systems or static or dynamic semantics of programming languages or calculi), referred to as *object systems*, into a fixed suitably chosen logic, referred to as the *meta logic*. HOAS prescribes that the object system be represented in the meta logic so that variable-binding, substitution and inference mechanisms of the former be captured by corresponding mechanisms of the latter.

Inspired by Church's idea to represent all logical connectives and quantifiers using λ-abstraction in simple type theory, HOAS originated more or less independently in [71, 111, 63, 108] and has ever since been extensively developed in frameworks with a wide variety of features and flavors. We can distinguish two main (overlapping) directions in these developments.

-**(I)** First, the employment of a chosen meta logic as a *pure logical framework*, used for defining object systems for the purpose of reasoning *inside those systems*. A standard example is higher-order logic (HOL) as the meta logic and first-order logic (FOL) as the

[4] Although the recent paper [106] argues otherwise.

object system. Thanks to affinities between the mechanisms of these two logics, one obtains an *adequate encoding* of FOL in HOL by merely declaring in HOL types and constants and stating the FOL axioms and rules as HOL axioms – then the mechanisms for building FOL deductions (including substitution, instantiation, etc.) are already present in the meta logic, HOL.

-**(II)** Second, the employment of the meta-logic to reason *about* the represented object systems, i.e., to represent not only the object systems, but also (some of) their *meta-theory*. (E.g., cut elimination is a property *about* Gentzen-style FOL, not expressible in a standard HOAS-encoding of FOL into HOL.) While direction (I) has been quasi-saturated by the achievement of quasi-maximally convenient logical frameworks (such Edinburgh LF [63] and generic Isabelle [108]), this second direction undergoes these days a period of active research. We distinguish two main approaches here:

-**(II.a)** The *HOAS-tailored framework approach* [142, 140, 80, 85, 146, 50, 2, 29, 113]. This is characterized by the extension of the pure logical frameworks as in (I) with meta-reasoning capabilities. The diad (object system, meta logic) from (I) becomes a triad: *object system*, *logical framework* where this system is specified, *meta-logical framework* where one can reason *about* the logical framework [110]. The challenge here is choosing suitable logical and meta-logical frameworks that allow for adequate HOAS encodings, as well as enough expressive meta-theoretic power. (The logical framework is typically chosen to be a weak logic, e.g., an intuitionistic logic or type system as in (I), or linear logic.)

Somewhat complementary to the above work on HOAS-tailored *meta-reasoning*, [141, 40] developed HOAS-tailored *recursive definition* principles in a logical framework distinguishing between a parametric and a primitive-recursive function space.

-**(II.b)** The *general-purpose framework approach* [39, 38, 13]. This approach employs a general-purpose setting for developing mathematics, such as ZF Set Theory, Calculus of Constructions, or HOL with Infinity, as the logical framework, with object-level bindings captured again by means of meta-level bindings, here typically functional bindings – this means that terms with bindings from the object system are denoted using *standard functions*. Here there is no need for the three-level architecture as in (II.a), since the chosen logical framework is already strong enough, meta-theoretic expressiveness not being a problem. However, the difficulty here is brought be the meta-level function space being wider than desired, containing so-called "exotic terms". Even after the function spaces are cut down to valid terms, adequacy is harder to prove than at (II.a), precisely because of the logic's expressiveness.

1.2.3 Our own view on syntax representation and reasoning, in a nutshell

We take a behavioral view, considering that the particular means of representing syntax are only important through their outcome: the generality and usefulness of the resulted definition and proof principles.

Indeed, all approaches, some of them admittedly more elegant than others, aim at capturing adequately the same Platonic concept of syntax. The actual route of achieving this is, in our opinion, not important in itself; however, it may suggest insightful definition and proof principles, nothing though that cannot be imported into a formalization based on any other approach (at least any based on a general-purpose framework)! Our own "implementation" of terms with bindings is a standard one based on α-classes, on top of which we define FOAS and HOAS machineries incorporating our own theoretical results, as well as borrowing what we consider to be useful consequences of other approaches.

Above, by definition and proof principles, we mainly mean:
- principles of recursive definitions, on the structure of syntax, for functions (a.k.a. structural recursion);
- principles of inductive reasoning, on the structure of syntax (a.k.a. structural induction);
- principles of definition and reasoning for inductively defined relations (a.k.a. rule induction).

Consequently, recursion and induction specific to (and convenient for) syntax with bindings, coming in both FOAS and HOAS flavors, will be the main themes of the first part of this dissertation.

1.2.4 Process algebra, coalgebra and behavioral logic

Process algebra debuted with Milner's Calculus of Communicating Systems (CCS) [87] and has ever since undergone a rapid development, receiving much attention from theoretical computer scientists, notably by the Dutch school led by Jan Bergstra, as reported in the monograph [17]. The inclusion of value-passing and channel-passing features (the latter brought by Milner's π-calculus [88]), have placed process algebra closer to real programming languages, by the possibility to express complex behavior compactly and naturally [138].

Process algebra emphasizes the view that the behavior of processes (and programs) is best described not by *inductive* means (like syntax is), but by the dual, *coinductive* means. The notion of bisimilarity as process equivalence and the associated coinduction proof method are central in process algebra. Work has been done in order to establish criteria for bisimilarity to be well-behaved, allowing for powerful versions of coinduction proof rules. Some of these criteria were stated for particular process calculi [138], while others were stated generally, depending on syntactic conditions on the format of the SOS rules that describe the process transitions – the latter pertain to the meta-theory of SOS specifications and are reviewed in

the recent monograph [97].

In some cases (for systems with bisimilarity being a congruence and with additional modularity properties), one can consider the notion of abstract process behavior independently of the syntax of processes, and then map processes to their denotation behaviors – this is achieved by regarding transition systems as *coalgebras* [134] and modeling the universe of behaviors as the *final coalgebra* (the most general cases being treated in [135] and [147]). Working with processes up to bisimilarity and working in the final coalgebra of behaviors are two faces of the same coin (both guaranteing modular reasoning), but sometimes the coalgebraic view brings more insight to the situation.

Yet another framework, developed to a large extent independently from the coalgebra theory, is *hidden algebra* and *behavioral logic* [56, 132], an extension of universal algebra with hidden sorts and with a notion of *behavioral equivalence* that identifies two items of a hidden sort if they yield the same results under all "experiments" of visible sort. A powerful idea developed in this setting is that of *incremental construction* of the desired relation as a coinductive argument, presented as *circular reasoning* [55] and recently (re)implemented as the theorem prover CIRC [77]. Since this setting involves equations with open terms (i.e., terms with variables), the approach has certain similarities with those from process algebra based on generic bisimilarity [37, 129, 26].

Our work can be regarded as an extension of the incremental-coinduction approach to cope with concurrent systems and eventually with variable bindings as well, and an Isabelle/HOL formalization of this approach. We have preferred to use a theorem prover for the formalization instead of a rewrite engine such as Maude (as was done in [77]) for two reasons:

- First, because we found that, in many concurrent process calculi (and also in concurrent programming languages), the incremental unfolding of the processes yields non-trivial first-order (or higher-order) side-conditions resulting either from the consideration of a more complex notion of equivalence such as weak bisimilarity or from the composition of simpler local side-conditions; many of these side-conditions can be automatically discharged by a tool for automatic first-order reasoning such as Isabelle's "blast" method.
- Second, for eventually being able to integrate coinduction into our planned overall package for representing and reasoning about syntax and behavior.

1.3 Conventions, notations and pointers to supporting scripts

We refer to all the involved collections as *sets*. We employ the lambda-abstraction symbol λ, standing for functional abstraction, as well as the standard logical connectives and quantifiers

($\neg$ for negation, $\wedge$ for conjunction, $\vee$ for disjunction, $\Longrightarrow$ for implication, $\Longleftrightarrow$ for equivalence (i.e., bi-implication), $\forall$ and $\exists$ for the universal and existential quantifiers), only in the meta-language of this paper, and *not* in the various formal languages that we discuss.

bool is the two-element set of booleans, $\{\mathsf{True}, \mathsf{False}\}$. $I\!N$, usually ranged over by m, n, i, is the set of natural (i.e., non-negative integer) numbers. $\mathsf{max}\ m\ n$ is the maximum between the numbers m and n. $A \times B$ is the cartesian product of A and B. $A \cup B$ is the union of A and B. Given a set of sets $\mathcal{A}$, $\bigcup\ \mathcal{A}$ is the union of all the elements of $\mathcal{A}$. $A \to B$ is the A-to-B function space.

$\mathbf{P}(A)$, $\mathbf{P}_{\neq\emptyset}(A)$ and $\mathbf{P}_f(A)$ are the A-powerset (i.e., the set of all subsets of A), the set of non-empty subsets of A, and the set of finite subsets of A, respectively. $\emptyset$ is the empty set. $\mathbf{List}(A)$ is the set of lists of elements from A. Nil or $[]$ denotes the empty list. When we consider typing contexts, which are lists of pairs of items, infixed "," denotes list concatenation. When thinking of lists as words, ϵ denotes the empty list and infixed $\#$ denotes list concatenation.

We mostly consider operations and relations in the curried form, and mostly represent relations as maps to **bool**. E.g., $\mathsf{App} : \mathbf{term} \to \mathbf{term} \to \mathbf{term}$ is a binary operation, and $\mathsf{fresh} : \mathbf{var} \to \mathbf{term} \to \mathbf{bool}$ is a relation between variables and terms. For a (binary) relation $R : A \to A \to \mathbf{bool}$, R^* is its reflexive-transitive closure. Given an n-ary relation R on A and $a_1, \ldots, a_n \in A$, we say that "$R\ a_1\ \ldots\ a_n$ holds" if $R\ a_1\ \ldots\ a_n = \mathsf{True}$. However, sometimes we regard relations as subsets of powersets rather than as maps to **bool**.

The mathematical statements (theorems, propositions, lemmas and corollaries) are numbered using a common counter, and their number is prefixed by the number of the chapter. For instance, Lemma 3.5 would indicate the 5th mathematical fact from Chapter 3, which would happen to be a lemma. We only give sketches of proofs for the various stated facts. The reader interested in more details about the proofs is referred to our Isabelle formalization – details on the connection between this dissertation's presentation and the corresponding formalization are given toward the end of each chapter. We refer to both sections or subsections as "sections", e.g., "Section 2.3" refers to a section, and "Section 2.3.1" refers to a subsection.

Most of the constructions and theorems presented in this dissertation have been formalized in Isabelle/HOL [103]. The formal scripts can be found at:
- [120, 121], for Chapter 2;
- [119], for Chapter 3;
- [122], for Chapter 4.

1.4 Technical preliminaries

In this section, we discuss technical prerequisites pertaining to syntax with bindings and Horn theory.

1.4.1 Syntax with bindings

In this dissertation, we consider syntax with bindings under static scoping (meaning variable-capture is not allowed in the substitution process). We assume from the reader familiarity with the basics of syntax with bindings, such as α-equivalence and substitution. Below, we briefly recall basic definitions pertaining to the paradigmatic syntax with bindings, namely, the untyped λ-calculus. For more details, we refer the reader to [20] or [74]. These definitions extend in the obvious way to any (possibly many-sorted) arbitrary syntax with bindings, specified by indicating:
- the syntactic categories (i.e., the sorts);
- the constructors, together with an indication of whether their arguments are free or variable-bound.

We fix an infinite set **var**, of variables, ranged over by x, y, z. We also fix an arbitrary map $\mathsf{pickDistinct} : \mathbf{P}_f(\mathbf{var}) \to \mathbf{var}$ (intended to pick a variable not belonging to a given set of variables), with the property that $\forall\, V \in \mathbf{P}_f(\mathbf{var}).\ \mathsf{pickDistinct}\ V \notin V$. (Such a map obviously exists, by the infiniteness of **var**.)

Let **qTerm**, the set of λ-calculus *quasi-terms*, ranged over by P, Q, be given by the following grammar:

$$P \quad ::= \quad \mathsf{qVar}\ x \mid \mathsf{qApp}\ P\ Q \mid \mathsf{qLm}\ x\ P$$

where we think of qVar as the injection of variables into quasi-terms, of qApp as application and of qLm as λ-abstraction; therefore, we think of x as intended to be *bound* in P within $\mathsf{qLm}\ x\ P$.

We define the relation $\mathsf{qFresh} : \mathbf{var} \to \mathbf{qTerm} \to \mathbf{bool}$ (where $\mathsf{qFresh}\ x\ P$ says "x is fresh for, i.e., does not appear free in, P"), by standard induction:
- $x \neq z \implies \mathsf{qFresh}\ z\ (\mathsf{qVar}\ x)$;
- $\mathsf{qFresh}\ z\ P\ \wedge\ \mathsf{qFresh}\ z\ Q \implies \mathsf{qFresh}\ z\ (\mathsf{qApp}\ P\ Q)$;
- $(x = z\ \vee\ \mathsf{qFresh}\ z\ P) \implies \mathsf{qFresh}\ z\ (\mathsf{qLm}\ x\ P)$.

One can easily see that, for all P, the set $\{x.\ \neg\ \mathsf{qFresh}\ x\ P\}$ is finite, which allows us to define $\mathsf{pickFresh} : \mathbf{P}_f(\mathbf{var}) \to \mathbf{P}_f(\mathbf{qTerm}) \to \mathbf{var}$ by $\mathsf{pickFresh}\ V\ W = \mathsf{pickDistinct}\ (V \cup \{x.\ \exists P \in W.\ \neg\ \mathsf{qFresh}\ z\ P\})$ and be sure that the following hold:
- $\mathsf{pickFresh}\ V\ W \notin V$;
- $\forall P \in W.\ \mathsf{qFresh}\ (\mathsf{pickFresh}\ V\ W)\ P$.

We define the depth (a.k.a. height) operator, $\mathsf{qDepth} : \mathbf{qTerm} \to I\!N$, by standard

recursion:
- $\mathsf{qDepth}\ (\mathsf{qVar}\ x) = 1$;
- $\mathsf{qDepth}\ (\mathsf{qApp}\ P\ Q) = \mathsf{max}\ (\mathsf{qDepth}\ P)\ (\mathsf{qDepth}\ Q)$;
- $\mathsf{qDepth}\ (\mathsf{qLm}\ x\ P) = 1 + \mathsf{qDepth}\ P$.

We define the variable-swapping operator $\_[\_\wedge\_]^v : \mathbf{var} \to \mathbf{var} \to \mathbf{var} \to \mathbf{var}$ by

$$x[z1 \wedge z_2]^v = \begin{cases} z_2, & \text{if } x = z_1 \\ z_1, & \text{if } x = z_2 \\ x, & \text{otherwise.} \end{cases}$$

We define the swapping operator $\_[\_\wedge\_]^q : \mathbf{qTerm} \to \mathbf{var} \to \mathbf{var} \to \mathbf{qTerm}$, where $P[z_1 \wedge z_2]^q$ is the quasi-term obtained from P by swapping z_1 with z_2 everywhere in P, by standard recursion:
- $(\mathsf{qVar}\ x)[z1 \wedge z_2]^q = \mathsf{qVar}\ (x[z1 \wedge z_2]^v)$;
- $(\mathsf{qApp}\ P\ Q)[z_1 \wedge z_2]^q = \mathsf{qApp}\ (P[z_1 \wedge z_2]^q)\ (Q[z_1 \wedge z_2]^q)$;
- $(\mathsf{qLm}\ x\ P)[z_1 \wedge z_2]^q = \mathsf{qLm}\ (x[z1 \wedge z_2]^v)(P[z_1 \wedge z_2]^q)$.

We define the substitution operator $\_[\_/\_]^q : \mathbf{qTerm} \to \mathbf{qTerm} \to \mathbf{var} \to \mathbf{qTerm}$, where $P[R/z]^q$ is the quasi-term obtained from P by substituting R for each free occurrence of z in P *in a capture-avoiding fashion*, by standard recursion:
- $(\mathsf{qVar}\ x)[R/z]^q = \begin{cases} R, & \text{if } x = z \\ \mathsf{qVar}\ x, & \text{otherwise;} \end{cases}$
- $(\mathsf{qApp}\ P\ Q)[R/z]^q = \mathsf{qApp}\ (P[R/z]^q)\ (Q[R/z]^q)$;
- $(\mathsf{qLm}\ x\ P)[R/z]^q = \mathsf{qLm}\ x'\ (P[x' \wedge x]^q[R/z]^q)$,

where $x' = \mathsf{pickFresh}\ \{z\}\ \{P\}$.

The α-*equivalence* (a.k.a. *naming equivalence*) relation $\simeq_\alpha$: $\mathbf{qTerm} \to \mathbf{qTerm} \to \mathbf{bool}$, can be standardly defined either inductively (as a relation) or recursively (as an operator) in a variety of equivalent ways, the most traditional of which being the following inductive one:
- (1) $\mathsf{qVar}\ x \simeq_\alpha \mathsf{qVar}\ x$;
- (2) $P \simeq_\alpha P' \ \wedge\ Q \simeq_\alpha Q' \Longrightarrow \mathsf{qApp}\ P\ Q \simeq_\alpha \mathsf{qApp}\ P'\ Q'$;
- (3) $\mathsf{qFresh}\ y\ P \ \wedge\ \mathsf{qFresh}\ y\ P' \ \wedge\ P[(\mathsf{qVar}\ y)/x]^q \simeq_\alpha P'[(\mathsf{qVar}\ y)/x']^q$
$\Longrightarrow \mathsf{qLm}\ x\ P \simeq_\alpha \mathsf{qLm}\ x'\ P'$.

Notice the special, "non-injective" treatment of the binding operator Lm. Equivalent definitions include those obtained from the above by one of the folowing variations of clause (3):
- A maximal-precondition variation:

— (3') $(\forall\ y.\ P[(\mathsf{qVar}\ y)/x]^q \simeq_\alpha P'[(\mathsf{qVar}\ y)/x']^q) \Longrightarrow \mathsf{qLm}\ x\ P \simeq_\alpha \mathsf{qLm}\ x'\ P'$;

Minimal-precondition and maximal precondition variations based on swapping (advocated by Nominal Logic [115]):

— (3") $\mathsf{qFresh}\ y\ P \ \wedge\ \mathsf{qFresh}\ y\ P' \ \wedge\ P[y \wedge x]^q \simeq_\alpha P'[y \wedge x']^q \Longrightarrow \mathsf{qLm}\ x\ P \simeq_\alpha \mathsf{qLm}\ x'\ P'$;

— (3"') $(\forall y.\ P[y \wedge x]^q \simeq_\alpha P'[y \wedge x']^q) \Longrightarrow \mathsf{qLm}\ x\ P \simeq_\alpha \mathsf{qLm}\ x'\ P'$.

It is well-known that $\simeq_\alpha$ is indeed an equivalence and, moreover, compatible with the syntactic constructs, freshness, depth, swapping and substitution. Let **term**, ranged over by X, Y, Z, be the set of α-equivalence classes of quasi-terms, which we call *terms*. For a quasi-term P, we write $\overline{P}$ for the term that is its α-equivalence class. Operators on terms corresponding to the quasi-term syntactic constructs,

- $\mathsf{Var} : \mathbf{var} \to \mathbf{term}$,
- $\mathsf{App} : \mathbf{term} \to \mathbf{term} \to \mathbf{term}$,
- $\mathsf{Lm} : \mathbf{var} \to \mathbf{term} \to \mathbf{term}$,
- $\mathsf{depth} : \mathbf{term} \to I\!N$,
- $\mathsf{fresh} : \mathbf{var} \to \mathbf{term} \to \mathbf{bool}$,
- $\_[\_ \wedge \_] : \mathbf{term} \to \mathbf{var} \to \mathbf{var} \to \mathbf{term}$,
- $\_[\_/\_] : \mathbf{term} \to \mathbf{term} \to \mathbf{var} \to \mathbf{term}$,

are defined standardly (knowing that $\simeq_\alpha$ is compatible with everything):

- $\mathsf{Var}\ x = \overline{\mathsf{qVar}\ x}$;
- $\mathsf{App}\ X\ Y = \overline{\mathsf{qApp}\ P\ Q}$, for some P and Q such that $X = \overline{P}$ and $Y = \overline{Q}$;
- $\mathsf{Lm}\ x\ X = \overline{\mathsf{qLm}\ x\ P}$, for some P such that $X = \overline{P}$;
- $\mathsf{depth}\ X = \mathsf{qDepth}\ P$ for some P such that $X = \overline{P}$;
- $\mathsf{fresh}\ x\ X = \mathsf{qFresh}\ x\ P$ for some P such that $X = \overline{P}$;
- $X[z_1 \wedge z_2] = \overline{P[z_1 \wedge z_2]^q}$, for some P such that $X = \overline{P}$;
- $X[Y/y] = \overline{P[R/y]^q}$, for some P and R such that $X = \overline{P}$ and $Y = \overline{R}$.

(Above, each time the choice of "some" quasi-term is immaterial.)

The set **term** therefore consists of the paradigmatic terms with bindings of the (untyped) λ-calculus, considered "up to α-equivalence". A brief way to indicate this set and its operators is the following:

"The set **term** is given by the following grammar:

$$X \ ::= \ \mathsf{Var}\ x \mid \mathsf{App}\ X\ Y \mid \mathsf{Lm}\ x\ X$$

where, within $\mathsf{Lm}\ x\ X$, any occurrence of x in X is assumed bound."

The implicit assumptions are that terms are identified modulo the notion of α-equivalence standardly induced by the above bindings and that freshness, swapping and substitution are the standard ones. We usually omit spelling the variable-injection operator x, writing x for the term $\mathsf{Var}\ x$. This is how we shall proceed with any other syntax with bindings considered in this dissertation.

1.4.2 Horn theory

Standard Horn theory

We fix an infinite set **FOvar**, ranged over by u, v, w, of *first-order variables* (*FO-variables*, for short). We also fix an FO-signature $\Sigma = (F, R)$, where:
- $F = (F_n)_{n \in \mathbb{N}}$ is a family of ranked sets of operation symbols;
- $R = (R_n)_{n \in \mathbb{N}}$ is a family of ranked sets of relation symbols.

We shall loosely write F instead of $\bigcup_{n \in \mathbb{N}} F_n$ and R instead of $\bigcup_{n \in \mathbb{N}} R_n$. To indicate operation and relation symbols of FO-signatures, we shall always use boxed (meta)symbols, such as $\boxed{\mathtt{f}}$ for an operation symbol and $\boxed{\mathtt{r}}$ for a relation symbol. Operation symbols of rank 0 (i.e., elements of F_0) are called *constant symbols*.

The set **FOterm**, of *FO-terms*, ranged over by t, is defined inductively as usual:
- any FO-variable u is an FO-term;
- given $n \in \mathbb{N}$, FO-terms $t_1, \ldots, t_n$ and $\boxed{\mathtt{f}} \in F_n$, $\boxed{\mathtt{f}}\ t_1\ \ldots\ t_n$ is again an FO-term.

An *atomic formula* (*atom*, for short) a is either a *formal equality* $t \equiv t'$ (i.e., a pair (t, t') for which we use a special notation) or a *formal predicate atom* $\boxed{\mathtt{r}}\ t_1\ \ldots\ t_n$ where $\boxed{\mathtt{r}} \in R_n$. A *($\Sigma$-)Horn clause* has the form $(\&_{i=1}^{k} a_i) \Longrightarrow b$, with the a_i's and b atoms, i.e., a formal implication between a formal conjunction of atoms and an atom. (Thus, "$\&$" and "$\Longrightarrow$" stand for formal conjunction and implication in the (FOL) Horn language.) If $k = 0$, a Horn clause as above is merely an atom, b.

Σ-*models* (*models*, for short) are the usual first-order models. Namely, a model is a triple $(A, (\boxed{\mathtt{f}}^A)_{\boxed{\mathtt{f}} \in F}, (\boxed{\mathtt{r}}^A)_{\boxed{\mathtt{r}} \in R})$, where:
- A is a set, called the *carrier set* of the model;
- for all $n \in \mathbb{N}$ and $\boxed{\mathtt{f}} \in F_n$, $\boxed{\mathtt{f}}^A : \mathbf{Cur}(A^n \to A)$ is a (curried) n-ary operation on A;
- for all $n \in \mathbb{N}$ and $\boxed{\mathtt{r}} \in R_n$, $\boxed{\mathtt{r}}^A : \mathbf{Cur}(A^n \to \mathbf{bool})$ is a (curried) n-ary relation on A.

Above, $\mathbf{Cur}(A^n \to C)$ is the curried version of $A^n \to A$, namely, $A \to \ldots \to A \to C$ (with A repeated n times).

When the operations are understood, we may write A for the whole model. A *morphism* between two models A and B is a map $h : A \to B$ that commutes with the operations and preserves the relations, i.e., such that:
- for all $n \in \mathbb{N}$, $\boxed{\mathtt{f}} \in F_n$, and $a_1, \ldots, a_n \in A$, $h(\boxed{\mathtt{f}}^A\ a_1\ \ldots\ a_n) = \boxed{\mathtt{f}}^B\ (h\ a_1)\ \ldots\ (h\ a_n)$;

- for all $n \in I\!N$, $\boxed{\mathrm{r}} \in R_n$, and $a_1, \ldots, a_n \in A$, if $\boxed{\mathrm{r}}^A\ a_1\ \ldots\ a_n$ holds, then $\boxed{\mathrm{r}}^B\ (h\ a_1)\ \ldots\ (h\ a_n)$ also holds.

Notice that the notion of morphism treats relations not just like Boolean operations, but fundamentally different from the way it treats operations: the condition for operations is equality, corresponding to an "iff", while that for relations is an "if".

Satisfaction of a Horn clause by a model is the usual Tarskian satisfaction. Namely:
- First, given any model A and valuation $\rho : \mathbf{FOvar} \to A$, we define the map $A_\rho : \mathbf{FOterm} \to A$ by naturally extending ρ to terms, as follows:
— $A_\rho\ x = \rho\ x$;
— $A_\rho\ (\boxed{\mathrm{f}}\ t_1\ \ldots\ t_n) = \boxed{\mathrm{f}}^A\ (A_\rho\ t_1)\ \ldots\ (A_\rho\ t_1)$.
- Then we define satisfaction of an atom by a model via a valuation, $A \models_\rho a$, as follows:
— for formal equality atoms: $A \models_\rho t \equiv t'$ iff $A_\rho\ t = A_\rho\ t'$;
— for formal predicate atoms: $A \models_\rho \boxed{\mathrm{r}}\ t_1\ \ldots\ t_n$ iff $\boxed{\mathrm{r}}^A\ (A_\rho\ t_1)\ \ldots\ (A_\rho\ t_n)$.
- Finally, we define satisfaction of a Horn clause by a model as follows:
— $A \models ((\&_{i=1}^k a_i) \Longrightarrow b)$ iff $\forall \rho : \mathbf{FOvar} \to A.\ (\forall i \in \{1, \ldots, k\}.\ A \models_\rho a_i) \Rightarrow A \models_\rho b$.

A *Horn theory* $\mathcal{H}$ is a collection of Horn clauses. A model A satisfies a Horn theory $\mathcal{H}$, written $A \models \mathcal{H}$, iff it satisfies each of its clauses. A model A is $(\Sigma, \mathcal{H})$-initial iff the following hold:
- $A \models \mathcal{H}$;
- For all models B, if $B \models \mathcal{H}$, then there exists a unique morphism $h : A \to B$.

The following is a well-known result:

Prop 1.1 *Given a Horn theory $\mathcal{H}$, there exists a $(\Sigma, \mathcal{H})$-initial model, which is unique up to isomorphism.*

We let $I_{\Sigma,\mathcal{H}}$ denote the $(\Sigma, \mathcal{H})$-initial model.

Specifying/characterizing a data type as initial in a Horn theory yields standardly an iteration principle for that data type. This principle's "contract" reads as follows: give a Σ-model satisfying $\mathcal{H}$, and get back a compositional map (i.e., a morphism) h from $I_{\Sigma,\mathcal{H}}$ to A. Note that defining a model means essentially indicating the desired iterative behavior for h.

Full recursion

It is well-known that full recursion[5] can be reduced to iteration for absolutely free data types (such as lists). It turns out that the same is true for full recursion modulo a Horn theory, with a similar construction,[6] as we show below.

[5] In this dissertation, all the recursion principles we discuss are "primitive", reason for which we omit the adjective "primitive" – so what we call "full recursion" is often called "primitive recursion" in the literature; we use the adjective "full" to contrast it with the more restrictive variant given by iteration.

[6] This simple observation may be folklore, but we were not able to find a reference to it in the literature.

For the remainder of this section, we fix a signature Σ and a Horn theory $\mathcal{H}$, and write I for $I_{\Sigma,\mathcal{H}}$.

A *full-recursion model* (*GR-model for short*) is essentially an extension of a standard model with extra arguments belonging to the initial model. Namely, it is triple $(A, (\mathsf{f}^A)_{\mathsf{f}\in F}, (\mathsf{r}^A)_{\mathsf{r}\in R}$, where:

- A is a set, called the *carrier set* of the GR-model;
- for all $n \in \mathbb{N}$ and $\mathsf{f} \in F_n$, $\mathsf{f}^A : \mathbf{Cur}((I \times A)^n \to A)$ is a (curried) n-ary operation on A;
- for all $n \in \mathbb{N}$ and $\mathsf{r} \in R_n$, $\mathsf{r}^A : \mathbf{Cur}((I \times A)^n \to \mathbf{bool})$ is a (curried) n-ary relation on A.

Above, the **Cur** operator is assumed to curry everything, including all the occurrences of the products $I \times A$; e.g., if f is a binary operation symbol (i.e., $\mathsf{f} \in F_2$), then f^A has the type $I \to A \to I \to A \to A$. Therefore, each standard argument (in A) comes in pair with an argument from I.

Again, when the operations are understood, we may write A for the whole GR-model.

We shall only be interested in morphisms from I (as a standard model), to a GR-model, which we call GR-morphisms. Given a GR-model A, a *GR-morphism to* A is a map $h : I \to A$ such that the following hold:

- for all $n \in \mathbb{N}$, $\mathsf{f} \in F_n$, and $a_1, \ldots, a_n \in I$, $h(\mathsf{f}^A\ a_1\ \ldots\ a_n) = \mathsf{f}^B\ a_1\ (h\ a_1)\ \ldots\ a_n\ (h\ a_n)$;
- for all $n \in \mathbb{N}$, $\mathsf{r} \in R_n$, and $a_1, \ldots, a_n \in A$, if $\mathsf{r}^A\ a_1\ \ldots\ a_n$ holds, then $\mathsf{r}^B\ a_1\ (h\ a_1)\ \ldots\ a_n\ (h\ a_n)$ also holds.

Satisfaction of a Horn clause by a GR-model is defined similarly to that by a standard model, but taking into account the extra I-arguments too. Namely:

- First, note that the carrier of I is a quotient of **FOterm** (and in fact the model I is a quotient of the $I_{\Sigma,\emptyset}$, the absolutely initial model, whose carrier can be taken to be **FOterm**); let $\pi : \mathbf{FOterm} \to I$ be the quotient map.
- Then, given any GR-model A and valuation $\rho : \mathbf{FOvar} \to A$, we define the map $A_\rho : \mathbf{FOterm} \to A$ by naturally extending ρ to terms, as follows:
  - $A_\rho\ x = \rho\ x$;
  - $A_\rho\ (\mathsf{f}\ t_1\ \ldots\ t_n) = \mathsf{f}^A\ (\pi\ t_1)\ (A_\rho\ t_1)\ \ldots\ (\pi\ t_n)\ (A_\rho\ t_n)$.
- Then we define satisfaction of an atom by a GR-model via a valuation, $A \models_\rho a$, as follows:
  - for formal equality atoms: $A \models_\rho t \equiv t'$ iff $A_\rho\ t = A_\rho\ t'$;
  - for formal predicate atoms: $A \models_\rho \mathsf{r}\ t_1\ \ldots\ t_n$ iff $\mathsf{r}^A\ (\pi\ t_1)\ (A_\rho\ t_1)\ \ldots\ (\pi\ t_n)\ (A_\rho\ t_n)$.
- Finally, we define satisfaction of a Horn clause by a GR-model as follows:
  - $A \models ((\&_{i=1}^k a_i) \Longrightarrow b)$ iff $\forall \rho : \mathbf{FOvar} \to A.\ (\forall i \in \{1, \ldots, k\}.\ A \models_\rho a_i) \Rightarrow A \models_\rho b$.

A GR-model A satisfies a Horn theory $\mathcal{H}$, written $A \models \mathcal{H}$, iff it satisfies each of its clauses.

Prop 1.2 *Given a Horn-theory $\mathcal{H}$ and a GR-model B such that $B \models \mathcal{H}$, there exists a unique GR-morphism to B.*

Proof sketch: First, note that the notion of GR-model behaves very much like a product

of I with a standard model, except that the results of the operations ignore the I-component of the result. Moreover, valuations act on the I-component essentially as identity.

These remarks form the basis of the reduction of full recursion to iteration. Define the model C as follows:

- The carrier set is $C = I \times B$;
- $\mathbf{f}^C\ (a_1, b_1)\ \ldots\ (a_n, b_n) = (I^{\mathbf{f}}\ a_1\ \ldots\ a_n, \mathbf{f}^B\ a_1\ b_1\ \ldots\ a_n\ b_n)$;
- $\mathbf{r}^C\ (a_1, b_1)\ \ldots\ (a_n, b_n) = \mathbf{r}^B\ a_1\ b_1\ \ldots\ a_n\ b_n$.

Then one can easily check the following facts:

- (1) $C \models \mathcal{H}$;
- (2) Any morphism from I to C is, on the first component;
- (3) There exists a bijection between the set of GR-morphisms to A and the set of morphisms between I and C. (The proof od this requires fact (2).)

Facts (1), (3) and Prop. 1.1 now give the desired result. $\square$

Chapter 2

FOAS [1]

2.1 Introduction

The main actor of this chapter is the data type of λ-calculus terms, **term**, with its construct operators, Var, App and Lm (hence, under a "FOAS view"), and with freshness and substitution. (All the involved notations are introduced in Section 1.4.1.) Note that a data type is not merely a collection of values (as such, **term** would be just a countable set, which could be taken to be $\mathbb{N}$), but it is a collection of values together with a collection of operations on them. In what follows, we refer to the above type as $\mathbf{term}_{C,F,S}$ with the three indexes standing for "Construct", "Freshness" and "Substitution".

Our contribution is an improvement of the knowledge of this type (in a uniform way, extendible to any syntax with static bindings). The contribution is formalized in Isabelle and in general guided by theorem-proving goals.

But what does "knowing" a data type means?
- (1) Of course, it primarily means having a way to define or uniquely characterize it rigorously, if possible in a clean mathematical way.
- (2) However, it also means understanding its structure, and being able to reason about it and define functions on it, again as cleanly as possible.

While the first meaning may seem like a closed subject mathematically, it is not so, due to its proviso about cleanness. Indeed, the standard definition based on α-equivalence (which in fact comes in a myriad of "equally standard" equivalent variants) is not the cleanest definition, as, e.g., it makes several choices later shown to be irrelevant – this is particularly unpleasant when it comes to formalization. α-equivalence is often cited as an example of a "messy" congruence relation.

Concerning the first meaning, we give a very simple characterization of $\mathbf{term}_{C,F,S}$ as the initial model of a Horn theory that does not involve any auxiliary operations, and contains a minimal set of clauses representing natural and well-known facts for terms.

This characterization is, we believe, interesting in itself, because it shows the (quite surprising) fact that $\mathbf{term}_{C,F,S}$ is no special data type. Thus, it is a data type of the same

kind as those of finite sets and finite bags (multisets): one that can be specified by listing some conditional clauses involving equations and atomic relations, and then letting standard closure mechanisms do the rest (which is generating the type freely by these clauses). And this is done *without introducing any auxiliary operators*.

More importantly however, this characterization proves useful w.r.t. the second meaning (of "knowing" a data type), in that it enables a recursive definition principle featuring built-in substitution compositionality, applicable to a large class of situations.

While the desire for substitution compositionality seems like a rather restrictive/specialized desire (since a form of "substitution" then needs to be available on the target domain, which is to be organized as a Horn model for the signature of $\mathbf{term}_{C,F,S}$), it, in fact, turns out to be pervasive in syntax-with-bindings developments. Indeed, "substitutions" are everywhere, and, moreover, relating substitutions by a compositionality property is usually a main fact that needs to be proved about a recursively defined function. We illustrate this point by many examples taken from the literature on syntax with bindings.

Back to (1), we also give an internal characterization of $\mathbf{term}_{C,F,S}$ as a model of the aforementioned Horn theory, with additional properties. This latter characterization is hardly an exciting mathematical result, but its formalization opens up a useful methodology for creating isomorphic bridges between different formal representations/implementations of terms (having the characterization as a "canon").

Here is an overview of the rest of this chapter.

In Section 2.2, we recall some known facts about induction principles tailored toward handling bindings.

The next six sections contain our contribution to the theory of recursive definitions for syntax with bindings. In Section 2.3, as motivation for the development to come, we present two typical problems in formal reasoning: semantic interpretation and HOAS interpretation. In Section 2.4, as a warm-up, we illustrate our Horn-based approach for a very simple example: defining the cardinality of finite sets. In Section 2.5, we state our main result – a freshness and substitution based recursion principle – and show how it solves the problems with which we started. In Section 2.6, we give many other applications of our recursive principles. In Section 2.7, we discuss the aforementioned internal characterization of the term model. In Section 2.8, we present swapping-based variations of our recursion principle. Then Section 2.9 describes a generalization and a formalization of our results. Finally, Section 2.10 discusses related work.

All throughout this section, unless otherwise specified, x, y, z range over variables (the set $\mathbf{var}$) and X, Y, Z over λ-calculus terms (the set $\mathbf{term}$).

2.2 Induction

In this section, we briefly review the state of the art in FOAS induction for syntax with bindings. While not including any of our original contribution to the theory of syntax, the discussed principles are incorporated in our Isabelle formalization (presented in Section 2.9). The two main subtopics here are:
- structural induction, discussed in Section 2.2.1,
- and rule induction, discussed in Section 2.2.2.

2.2.1 Structural induction

Since they are "constructed", i.e., completely generated by the term constructs, the λ-calculus terms are already subject to general-purpose structural induction:

Prop 2.1 *To prove that* $\forall X.\ \psi\ X$ *holds, it suffices to prove the following, for all* x, X, Y*:*
- *(1)* $\psi\ x$*;*
- *(2)* $\psi\ X\ \wedge\ \psi\ Y \Longrightarrow \psi\ (\mathsf{App}\ X\ Y)$*;*
- *(3)* $\psi\ X \Longrightarrow \psi\ (\mathsf{Lm}\ x\ X)$*.*

Proof sketch. This follows from the structural induction principle for quasi-terms, together with the fact that terms are obtained from quotienting quasi-terms. □

Another (completely standard) general-purpose form of induction available for terms is the one based on depth:

Prop 2.2 *To prove that* $\forall X.\ \psi\ X$ *holds, it suffices to prove the following, for all* X*:*
$(\forall Y.\ \mathsf{depth}\ Y < \mathsf{depth}\ X \Longrightarrow \psi\ Y) \Longrightarrow \psi\ X$*.*

Proof sketch. Again, this follows from the corresponding fact from quasi-terms, together with the preservation of depth by α-equivalence.

Prop. 2.2 is clearly more general than Prop. 2.1, but, as usual with the tradeoff between generality and specificity, less convenient in concrete situations (in that, if Prop. 2.1 is applicable, it typically yields shorter and more readable proofs). A principle intermediate between the two, taking advantage of the abstract flexibility offered by Prop. 2.2 only when it is usually needed most – in the Lm case – is given below.

Define the operator $\mathsf{swapped} : \mathbf{term} \to \mathbf{P}(\mathbf{term})$ by
$\mathsf{swapped}\ X = \{X[z_1 \wedge z_1'] \ldots [z_n \wedge z_n'].\ n \in I\!N,\ z_1, z_1', \ldots, z_n, z_n' \in \mathbf{var}\}$.

Prop 2.3 *To prove that* $\forall X.\ \psi\ X$ *holds, it suffices to prove the following, for all* x, X, Y*:*
- *(1)* $\psi\ x$*;*
- *(2)* $\psi\ X\ \wedge\ \psi\ Y \Longrightarrow \psi\ (\mathsf{App}\ X\ Y)$*;*
- *(3)* $(\forall Y \in \mathsf{swapped}\ X.\ \psi\ Y) \Longrightarrow \psi\ (\mathsf{Lm}\ x\ X)$*.*

Proof sketch. Immediate from Prop. 2.2, given that $\forall\ Y \in \mathsf{swapped}\ X.\ \mathsf{depth}\ Y = \mathsf{depth}\ X < \mathsf{depth}(\mathsf{Lm}\ x\ X)$. $\square$

The above is all fine, but practice in reasoning about bindings has shown that these principles are not suitable/convenient for an important class of situations: those involving freshness assumptions in the Lm-case (case (3) above). Making such freshness assumptions is known as *Barendregt's variable convention*, stated in [20] on page 26: "If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g., definition, proof), then in these terms all bound variables are chosen to be different from the free variables."

It turns out that this convention can be partly made rigorous by objectifying the aforementioned proof context as an extra parameter for the predicate to be proved. This insight, due to Urban and Tasson [153] (see also [105]), yields the following improved induction principle:

Prop 2.4 *Let* **param***, ranged over by P, Q, be a set (whose elements we call* parameters*) and let* $\mathsf{varsOf} : \mathbf{param} \to \mathbf{P}_f(\mathbf{var})$ *be a function and* $\varphi : \mathbf{term} \to \mathbf{param} \to \mathbf{bool}$ *a predicate. Then, to prove that* $\forall X\ P.\ \varphi\ X\ P$ *holds, it suffices to prove the following, for all* x, X, Y, P*:*
- *(1)* $\varphi\ x\ P$*;*
- *(2)* $(\forall Q.\ \varphi\ X\ Q)\ \wedge\ (\forall Q.\ \varphi\ Y\ Q) \Longrightarrow \varphi\ (\mathsf{App}\ X\ Y)\ P$*;*
- *(3)* $x \notin \mathsf{varsOf}\ P\ \wedge\ (\forall Q.\ \varphi\ X\ Q) \Longrightarrow \varphi\ (\mathsf{Lm}\ x\ X)\ P$*.*

Proof sketch. Apply Prop. 2.3, taking $\psi = \lambda X.\ \forall P.\ \varphi\ X\ P$. For the only interesting case, namely the Lm-case, the proof goes as follows: We assume that $\forall Y \in \mathsf{swapped}\ X.\ \forall P.\ \varphi\ Y\ P$ and need to show $\varphi\ (\mathsf{Lm}\ x\ X)\ P$. Pick a variable y fresh which is completely fresh (i.e., $y \neq x$, $\mathsf{fresh}\ y\ X$ and $y \notin \mathsf{varsOf}\ P$), and let $Y = X[y \wedge x]$. Then we have $Y \in \mathsf{swapped}\ X$, and therefore, by the assumption, $\forall P.\ \varphi\ Y\ P$ holds. Hence, by point (3) and the choice of y, we have $\varphi\ (\mathsf{Lm}\ x\ X)\ P$. But, again by the choice of y, we have $\mathsf{Lm}\ y\ Y = \mathsf{Lm}\ x\ X$, and therefore $\varphi\ (\mathsf{Lm}\ x\ X)\ P$ holds, as desired. $\square$

The main twist of Prop. 2.4 is Case (3), where the considered bound variable x is assumed free "everywhere else", i.e., in the proof context packed as parameter P.

2.2.2 Rule induction

Let us consider a standard inductively defined relation, β-reduction, $\rightsquigarrow : \mathbf{term} \to \mathbf{term} \to \mathbf{bool}$, given by the following clauses:

$$\frac{\cdot}{\mathsf{App}\ (\mathsf{Lm}\ x\ Y)\ X \rightsquigarrow Y[X/x]}(\mathsf{Beta}) \qquad \frac{X \rightsquigarrow Y}{\mathsf{Lm}\ z\ X \rightsquigarrow \mathsf{Lm}\ z\ Y}(\mathsf{Xi})$$

$$\frac{X \rightsquigarrow Y}{\mathsf{App}\ X\ Z \rightsquigarrow \mathsf{App}\ Y\ Z}(\mathsf{AppL}) \qquad \frac{X \rightsquigarrow Y}{\mathsf{App}\ Z\ X \rightsquigarrow \mathsf{App}\ Z\ Y}(\mathsf{AppR})$$

The following discussion applies not only to this relation, but to a large class of inductively defined relations, as shown in [150] (see also [152]).

The default rule induction principle for $\rightsquigarrow$, stating that any relation $\theta : \mathbf{term} \to \mathbf{term} \to \mathbf{bool}$ satisfying the above clauses (with $\rightsquigarrow$ replaced by θ) includes $\rightsquigarrow$, is not enough in typical proofs, which again need to employ Barendregt's convention. The following is an improvement in this direction:

Prop 2.5 *Let* **param**, *ranged over by* P, Q, *and* varsOf *be as in Prop. 2.4, and let* $\varphi : \mathbf{term} \to \mathbf{term} \to \mathbf{param} \to \mathbf{bool}$. *Then, to prove that* $\forall X\ Y\ P.\ X \rightsquigarrow Y \Longrightarrow \varphi\ X\ Y\ P$, *it suffices to prove the following, for all* x, z, X, Y, Z, P*:*
- *(1)* $x \notin \mathsf{varsOf}\ P$ *implies* $\varphi\ (\mathsf{App}\ (\mathsf{Lm}\ x\ Y)\ X)\ (Y[X/x])\ P$*;*
- *(2)* $z \notin \mathsf{varsOf}\ P$ *and* $X \rightsquigarrow Y$ *and* $\forall Q.\ \varphi\ X\ Y\ Q$ *imply* $\varphi\ (\mathsf{Lm}\ z\ X)\ (\mathsf{Lm}\ z\ Y)\ P$*;*
- *(3)* $X \rightsquigarrow Y$ *and* $\forall Q.\ \varphi\ X\ Y\ Q$ *imply* $\varphi\ (\mathsf{App}\ X\ Z)\ (\mathsf{App}\ Y\ Z)\ P$*;*
- *(4)* $X \rightsquigarrow Y$ *and* $\forall Q.\ \varphi\ X\ Y\ Q$ *imply* $\varphi\ (\mathsf{App}\ Z\ X)\ (\mathsf{App}\ Z\ Y)\ P$.

Proof sketch. The next fact follows by standard rule induction on $\rightsquigarrow$ using properties (1)-(4):
$X \rightsquigarrow Y \Longrightarrow (\forall P, n, z_1, z_1', \ldots, z_n, z_n'.\ \varphi\ (X[z_1 \wedge z_1'] \ldots [z_n \wedge z_n'])\ (Y[z_1 \wedge z_1'] \ldots [z_n \wedge z_n'])\ P)$.

For the interesting cases, (1) and (2), the proof is based on the following properties of *the definitional clauses* of $\rightsquigarrow$, which form the basis for the generalization given in [150]:
- All the involved functions and side-conditions are *equivariant*, in that they commute with swapping.
- All the involved variables are distinct, and all variables bound anywhere in a clause are fresh for the conclusion of that clause. $\square$

2.2.3 Case analysis and inversion rules

Case analysis and rule inversion are essentially trivial forms of structural induction and rule induction, respectively. Consequently, "fresh" versions for them are available along the lines of the results from Sections 2.2.1 and 2.2.2.

In the next two propositions, **param**, ranged over by P, Q, and varsOf are as in Prop. 2.4.

Prop 2.6 (Fresh case analysis) *Let* $X \in \mathbf{term}$, $P \in \mathbf{param}$ *and* $\varphi \in \mathbf{bool}$.[2] *Then, to prove that* φ *holds, it suffices to prove the following:*
- $\forall x \in \mathbf{var}.\ X = x \Longrightarrow \varphi$*;*
- $\forall Y\, Z.\ X = \mathsf{App}\ Y\ Z \Longrightarrow \varphi$*;*
- $\forall y\ Y.\ y \notin \mathsf{varsOf}\ P\ \wedge\ X = \mathsf{Lm}\ y\ Y \Longrightarrow \varphi$.

[2] Typically, φ is a statement about X and P.

Proof sketch. Immediately, by Prop. 2.4. □

Prop 2.7 (Fresh rule inversion) *Let* $U, V \in \mathbf{term}$, $P \in \mathbf{param}$ *and* $\varphi \in \mathbf{bool}$.[3] *Assume* $U \rightsquigarrow V$. *Then, to prove that* φ *holds, it suffices to prove the following:*
- *(1)* $\forall x\ X\ Y.\ x \notin \mathsf{varsOf}\ P\ \wedge\ U = \mathsf{App}\ (\mathsf{Lm}\ x\ Y)\ X\ \wedge\ V = Y[X/x] \Longrightarrow \varphi$;
- *(2)* $\forall z\ X\ Y.\ z \notin \mathsf{varsOf}\ P\ \wedge\ U = \mathsf{Lm}\ z\ X\ \wedge\ V = \mathsf{Lm}\ z\ Y \Longrightarrow \varphi$;
- *(3)* $\forall X\ Y\ Z.\ U = \mathsf{App}\ X\ Z\ \wedge\ V = \mathsf{App}\ Y\ Z \Longrightarrow \varphi$;
- *(4)* $\forall X\ Y\ Z.\ U = \mathsf{App}\ Z\ X\ \wedge\ V = \mathsf{App}\ Z\ Y \Longrightarrow \varphi$.

Proof sketch. Immediately, by Prop. 2.5. □

2.3 Two problems of rigorous/formal reasoning

We now embark on the presentation of our contribution – FOAS recursion principles. In this section, we present two typical situations in formal reasoning about syntax with bindings – interpretation in a semantic domain and HOAS representation – as motivation for the desire to have convenient (conceptual and formal) tools for recursive definitions sensible to the mechanisms of binding *and substitution*.

2.3.1 Problem I – Interpretation of syntax in semantic domains

Suppose we want to interpret terms in a semantic domain D (ranged over by d) endowed with operators

- $\mathsf{APP} : D \to D \to D$,
- $\mathsf{LM} : (D \to D) \to D$,

by matching the syntactic App and Lm with the semantic APP and LM, mapping syntactic binders to functional binders. For this, we need the collection of valuations (of variables into the domain), $\mathbf{val} = (\mathbf{var} \to D)$, ranged over by ρ. Then, the definition of the interpretation, $[\_] : \mathbf{term} \to \mathbf{val} \to D$, needs to proceed recursively on the structure of terms:

- $[X]\ \rho = \rho\ x$,
- $[\mathsf{App}\ X\ Y]\ \rho = \mathsf{APP}\ ([X]\ \rho)\ ([Y]\ \rho)$,
- $[\mathsf{Lm}\ x\ X]\ \rho = \mathsf{LM}\ (\lambda d : D.\ [X]\ (\rho[x \leftarrow d]))$,

where $\rho[x \leftarrow d]$ is the valuation ρ updated at x with d. Moreover, we usually wish to prove the following:

[3] Typically, φ is a statement about U, V and P.

- The obliviousness of the interpretation to the values of variables fresh for the represented term:
  $\mathsf{fresh}\ x\ X\ \wedge\ \rho =_x \rho' \Longrightarrow [X]\ \rho = [X]\ \rho'$,
  where $\rho =_x \rho'$, read "ρ equals ρ' everywhere but in x", is defined to be $\forall y \neq x.\ \rho\ y = \rho'\ y$.

- Compositionality w.r.t. substitution versus environment update, a.k.a. the *substitution lemma*, holds:
  $[Y[X/x]]\ \rho = [Y]\ (\rho[x \leftarrow [X]\ \rho])$.

Parenthesis. The above recursive definition and facts to be proved follow a canonical pattern of interpreting syntax in denotational semantics and formal logic. To give another well-known example, the semantics of FOL formulas φ is typically given as a relation $M \models_\rho \varphi$, read "M satisfies φ for the valuation $\rho : \mathbf{var} \to M$"; and the clause for the universal quantifier, which we denote by All,[4] is the following:

$$M \models_\rho \mathsf{All}\ x\ \varphi \quad \text{iff} \quad \forall m \in M.\ M \models_{\rho[x \leftarrow m]} \varphi.$$

Despite the different notation, this clause if of the same nature as that for Lm, in that the syntactic variable binding is captured in both cases by a semantic abstraction (functional in one case, universal in the other) in conjunction with environment update. Moreover, here (in FOL) we also wish to prove that the interpretation does not depend on the fresh variables and that the substitution lemma holds. In fact, most of the basic properties of FOL, including those necessary for the proof of the completeness theorem, rely on these two.

The problem and solution discussed below for λ-calculus applies equally to this FOL example and to many others in the same category.
End of parenthesis.

The problem with the above definition of $[\_]$ is that, at the Lm-clause, we have to show the choice of the representatives x and X in $\mathsf{Lm}\ x\ X$ immaterial. In other words, this clause proceeds as if Lm were a *free* construct on terms, while it is not (since there exist x, X, x', X' such that $(x, X) \neq (x', X')$ and $\mathsf{Lm}\ x\ X = \mathsf{Lm}\ x'\ X'$). Here is another way, perhaps the most rigorous one, to express the problem: if we replace, in the definition, terms with quasi-terms, and the term constructs Var, App and Lm with the corresponding quasi-term constructs qVar, qApp and qLm, we do have a valid definition of an operator on quasi-terms; however, to lift this to an operator on terms, we need to prove that it preserves α-equivalence. This problem is typically by-passed in informal texts, relying on "folklore", but needs to be faced and solved within a formalization.

[4] Remember our convention to use the symbols $\forall$ and λ only in the meta-language of dissertation.

The problem is actually more interesting and more challenging than may look at first. We urge the reader to try to solve it before reading our solution.

2.3.2 Problem II – Compositional HOAS representation

Recall that HOAS (Higher-Order Abstract Syntax) prescribes the representation of formal systems of interest, referred to as the *object systems*, into a fixed, but customizable *logical framework*, a.k.a. the *meta system*, by capturing object-level features such as binding mechanisms directly as corresponding meta-level features. While logical frameworks in current use are typically more complex (LF [63], intuitionistic HOL [108], etc.), here we restrict ourselves to the λ-calculus augmented with constants as the logical framework.[5] Again, the reader may easily notice that our discussion is rather general, and in particular it applies to the aforementioned more complex logical frameworks as well.

To augment the λ-calculus with constants, we fix **const**, ranged over by c, a set of (yet unspecified) *constants*. We extend the syntax of λ-calculus with these constants as usual, so that the new terms, which we call **const**-terms and whose set we denote by **term**(**const**), are now:
- either (injections of) variables x, or applications $\mathsf{App}\ X\ Y$, or λ-abstractions $\mathsf{Lm}\ x\ X$ (as before),
- or injections of constants, $\mathsf{Ct}\ c$.

Constants do not participate in bindings and are unaffected by substitution. Just like for variables, we usually omit spelling the constant-injection Ct, writing c instead of $\mathsf{Ct}\ c$.

We take **term**(**const**) as our logical framework. (A complete logical framework, of course, also has judgement mechanisms, typically given by a type system in combination with a reduction relation; for instance, in our case, reduction may be β or $\beta\eta$-reduction. In this section, we ignore the largely orthogonal judgement aspect of HOAS and consider the syntax representation aspect only.)

Say our object system is the λ-calculus itself (without constants). Its natural HOAS representation into **term**(**const**) proceeds as follows:

- Instantiate **const** to a two-element set, $\{\mathsf{APP}, \mathsf{LM}\}$, to match the two syntactic constructs App and Lm. (Note that one does not explicitly represent object-level variables and their injection into terms, as they will be captured directly by the meta-level items.)

- Define the representation map $\mathsf{rep} : \mathbf{term} \to \mathbf{term}(\{\mathsf{APP}, \mathsf{LM}\})$ (with postfix superscript notation) recursively on the structure of terms:
  — (1) $\mathsf{rep}\ x = x$,

[5] Already this simple λ-calculus with constants is a fairly standard HOAS framework; it is used in several places in the literature, including in [14] and as part of the Hybrid HOAS framework in [13, 95].

— (2) $\mathsf{rep}(\mathsf{App}\ X\ Y) = \mathsf{App}\ (\mathsf{App}\ \mathsf{APP}\ (\mathsf{rep}\ X))\ (\mathsf{rep}\ Y)$,
— (3) $\mathsf{rep}(\mathsf{Lm}\ x\ X) = \mathsf{App}\ \mathsf{APP}\ (\mathsf{Lm}\ x\ (\mathsf{rep}\ X))$.

In the more standard notation for λ-terms (which we avoid in order to prevent confusion with the meta-language of this dissertation), this is how the above equations would look like:

- $\mathsf{rep}\ x = x$,
- $\mathsf{rep}(X\ Y) = \mathsf{APP}\ (\mathsf{rep}\ X)\ (\mathsf{rep}\ Y)$,
- $\mathsf{rep}(\lambda\ x.\ X) = \mathsf{LM}\ (\lambda\ x.\ (\mathsf{rep}\ X))$.

Part of what makes a HOAS encoding very convenient is its *compositionality* w.r.t. substitution:
- (4) $\mathsf{rep}(Y[X/x]) = (\mathsf{rep}\ Y)[(\mathsf{rep}\ X)/x]$,
and indeed this is the main fact that is typically proved informally, by "pen-and-paper", about a HOAS representation. Preservation of freshness is another fact that turns out to be useful (see, e.g., [64], page 657), even though this is seldom acknowledged explicitly:
- (5) $\mathsf{fresh}\ x\ X \Longrightarrow \mathsf{fresh}\ x\ (\mathsf{rep}\ X)$.

As before, the problem with the definition of rep and with the (usual) informal proof of compositionality is the non-rigorous treatment of the Lm-case.

2.4 Intermezzo – solving a genuinely "ordinary" problem

We have claimed in the introduction of this chapter that terms with bindings are no special data types. Treating them "ordinarily" is the key to our proposed solution to Problems I and II. As a warm-up, we consider the case of a non-trivial, but "ordinary" data type: finite sets over a fixed universe of items. But first, we revisit the most ordinary situation of all – that of the absolutely free constructs on lists.

For the rest of this section, we fix a set **item**, ranged over by i, j, whose elements we call *items*.

2.4.1 Recursion on lists

Recall that **List**(**item**), which we let be ranged over by l, is the set of lists of items. Consider the following standard definition of the list length:

- $\mathsf{length}\ \mathsf{Nil} = 0$,
- $\mathsf{length}\ (\mathsf{Cons}\ i\ l) = \mathsf{length}\ l + 1$

The above is a valid iterative definition of a map $\mathsf{length} : \mathbf{List}(\mathbf{item}) \to I\!N$. A well-known alternative way to regard this definition is through the *initiality of the algebra of lists*, as follows. Consider the algebraic signature Σ_L consisting of:

- A constant symbol, $\boxed{\mathsf{Nil}}$;
- For each $i \in \mathbf{item}$, a unary operation symbol, $\boxed{\mathsf{Cons}_i}$.

(Thus, the signature has one constant symbol and a number of unary operation symbols equal to the number of items.) Then $\mathbf{List}(\mathbf{item})$ becomes a Σ_L-algebra by interpreting $\boxed{\mathsf{Nil}}$ as Nil and each $\boxed{\mathsf{Cons}_i}$ as $\mathsf{Cons}\ i$. In fact, $\mathbf{List}(\mathbf{item})$ is the *initial* Σ_L*-algebra* (in the notation of Section 1.4.2, the initial $(\Sigma_L, \emptyset)$-algebra), meaning that, for all Σ_L-algebras $(A, \boxed{\mathsf{Nil}}^A, (\boxed{\mathsf{Cons}_i}^A)_{i \in \mathbf{item}})$, there exists a unique morphism from $\mathbf{List}(\mathbf{item})$ to A, i.e., a unique map $H : \mathbf{List}(\mathbf{item}) \to A$ that commutes with the operations:

- $H\ \mathsf{Nil} = \boxed{\mathsf{Nil}}^A$,
- $H\ (\mathsf{Cons}\ i\ l) = \boxed{\mathsf{Cons}_i}^A (H\ l)$ for all $i \in \mathbf{item}$ and $l \in \mathbf{List}(\mathbf{item})$.

In this light, the above definition of length can be viewed as (given by) a definition of a Σ_L-algebra structure on $I\!N$, namely:

- $\boxed{\mathsf{Nil}}^{I\!N} = 0$,
- $\boxed{\mathsf{Cons}_i}^{I\!N} n = n + 1$ for all $i \in \mathbf{item}$ and $n \in I\!N$,

and then taking length to be the unique Σ_L-algebra morphism from $\mathbf{List}(\mathbf{item})$ to $I\!N$.

Summing up: *to define a map from the set of lists to another set, it suffices to endow the latter set with list-like operations* (i.e., to some extent, to treat the latter set as if consisting of "generalized lists").

2.4.2 Recursion on finite sets

Recall that $\mathbf{P}_f(\mathbf{item})$, which we let be ranged over by s, is the set of finite sets of items. A typical recursive definition of the operator $\mathsf{card} : \mathbf{P}_f(\mathbf{item}) \to I\!N$, giving the cardinal of a finite set, goes as follows:

- $\mathsf{card}\ \emptyset = 0$,
- $\mathsf{card}\ (\{i\} \cup s) = \mathsf{card}\ s + 1$, if $i \notin s$.

To have this definition justified rigorously, we need to show that, at the second clause, the choice of the representatives i and s for $\{i\} \cup s$ is irrelevant. This of course can be easily

worked around, but we are after a sufficiently general solution, that attacks the source of the difficulty in general terms.

The most apparent difficulty is that the invoked constructs in the clauses are not free (here, the operator sending (i, s) to $\{i\} \cup s$ is not injective). But even if they *were* free, we would have a second difficulty: the condition "$i \notin s$" of the second clause would destroy case-completeness (i.e., the definitional clauses would not cover all cases). Intuitively though, these two difficulties are also two features supposed to work in tandem to resolve each other.

On our way towards a solution, we note that switching perspectives from recursion to initiality leads to a powerful generalization in the style of algebraic specifications: *although the constructors may not be free, our data type may still be initial amongst all models of a Horn theory; hence an iteration principle may still be available.* (See Section 1.4.2.)

Unfortunately however, unlike for the free case, there may be many natural ways in which a non-free data type could be characterized as initial, corresponding to different choices of its "primitive" operations and relations (hence of the signature) *and of the characterizing clauses*. Coming up with characterizations able to define functions of interest as the initial morphisms therefore requires both mathematical creativity and experience with the use of that data type, but can also be partly guided by investigating "desired" recursive equations for given examples, as we shall explain shortly.

For instance, it is well-known that $\mathbf{P}_f(\mathbf{item})$ together with $\emptyset$ and $\cup$ is the free semi-lattice over **item**, or, equivalently, it is initial among all models over the signature of $\emptyset$, $\cup$ and the singletons $\{i\}$ with $i \in \mathbf{item}$. This indeed yields an iteration principle: to define a map $H : \mathbf{P}_f(\mathbf{item}) \to A$,

- endow A with operations $\boxed{\mathsf{Emp}}^A \in A$, $\boxed{\mathsf{Un}}^A : A \to A \to A$ and $\boxed{\mathsf{Singl}_i}^A \in A$ for all $i \in \mathbf{item}$;
- show that $(A, \boxed{\mathsf{Emp}}^A, \boxed{\mathsf{Un}}^A)$ is a semi-lattice, i.e., that $\boxed{\mathsf{Un}}^A$ is associative, commutative and idempotent and has $\boxed{\mathsf{Emp}}^A$ as neutral element;
- take H to be the unique map that commutes with the operators, in that
  - $H\ \emptyset = \boxed{\mathsf{Emp}}^A$,
  - $H\ (s \cup s') = \boxed{\mathsf{Un}}^A\ (H\ s)\ (H\ s')$,
  - $H\ \{i\} = \boxed{\mathsf{Singl}_i}^A$ for all $i \in \mathbf{item}$.

Thus, for the non-free case, while the end product is still an iteration principle, one needs to go further than writing the recursive equations and prove that the operators involved in these equations (here, $\boxed{\mathsf{Emp}}^A$, $\boxed{\mathsf{Un}}^A$ and $\boxed{\mathsf{Singl}_i}^A$) are well-behaved.

Of course, the initial semi-lattice characterization of $\mathbf{P}_f(\mathbf{item})$ does *not* help in justifying the above definition of card, since the latter involves operators different than those for

semi-lattices. The question here is: What are the signature and the Horn theory that make the definition of card go through? By analyzing the desired clauses for card, we first notice the involved operations and relations:

- the constant $\emptyset \in \mathbf{P}_f(\mathbf{item})$,
- a unary operation $(\{i\} \cup \_) : \mathbf{P}_f(\mathbf{item}) \to \mathbf{P}_f(\mathbf{item})$ for each $i \in \mathbf{item}$,
- a unary relation $(i \notin \_) : \mathbf{P}_f(\mathbf{item}) \to \mathbf{bool}$ for each $i \in \mathbf{item}$.

Thus, we are suggested the following signature Σ_S, consisting of operation and relation symbols with associated ranks:

- a constant symbol, $\boxed{\mathsf{Emp}}$;
- for each $i \in \mathbf{item}$, a unary operation symbol, $\boxed{\mathsf{Insert}_i}$;
- for each $i \in \mathbf{item}$, a unary relation symbol, $\boxed{\mathsf{fresh}_i}$.

$\mathbf{P}_f(\mathbf{item})$ becomes a Σ_S-model by interpreting $\boxed{\mathsf{Emp}}$ as $\emptyset$, $\boxed{\mathsf{Insert}_i}$ as $(\{i\} \cup \_)$, and $\boxed{\mathsf{fresh}_i}$ as $(i \notin \_)$.

Can we find a suitable Horn theory $\mathcal{H}_S$ for which our model $\mathbf{P}_f(\mathbf{item})$? Yes, here it is:

1. $\boxed{\mathsf{fresh}_i}\ \boxed{\mathsf{Emp}}$, for each $i \in \mathbf{item}$;[6]
2. $\boxed{\mathsf{fresh}_i}\ u \Longrightarrow \boxed{\mathsf{fresh}_i}\ (\boxed{\mathsf{Insert}_j}\ u)$, for each $i, j \in \mathbf{item}$ with $i \neq j$;
3. $\boxed{\mathsf{Insert}_i}(\boxed{\mathsf{Insert}_i}\ u) \equiv u$, for each $i \in \mathbf{item}$;
4. $\boxed{\mathsf{Insert}_i}(\boxed{\mathsf{Insert}_j}\ u) \equiv \boxed{\mathsf{Insert}_j}(\boxed{\mathsf{Insert}_i}\ u)$, for each $i, j \in \mathbf{item}$ with $i \neq j$;

Prop 2.8 $\mathbf{P}_f(\mathbf{item})$ *is the initial* $(\Sigma_S, \mathcal{H}_S)$*-model.*

Proof hint. The desired morphism can be defined as a relation and then showed to be functional. See [102], page 389-390, for a proof of a very similar nature. □

We came up with the above clauses essentially as follows:
- separating the data type "primitive" constructs, here the operators $\boxed{\mathsf{Emp}}$ and $(\{i\} \cup \_)$, from the "defined" operators and relations, here the relation $(i \notin \_)$
- choosing definition-like clauses for the "defined" items: clauses 1,2.
- treating the desired notion of equality as a "defined" item too, and therefore choosing definition-like clauses for it: clauses 3,4.

[6] That is to say, for each item i we consider one such Horn clause in the theory; and similarly below.

Now we are almost ready to justify the definition of card. One final obstacle is that iteration is not enough, full recursion being needed – this is because, in the desired recursive clause we started with,

$$\mathsf{card}\ (\{i\} \cup s) = \mathsf{card}\ s + 1, \text{ if } i \notin s,$$

for a fixed i, the definition of card on $\{i\} \cup s$ depends not only on $\mathsf{card}\ s$ (i.e., on the recursively computed result for the component), but also on s (i.e., on the component itself).

Here, what comes to our help is the Horn-based full recursion principle from Section 1.4.2 – indeed, the difficult part was to organize the source set $\mathbf{P}_f(\mathbf{item})$ as a suitable initial model, since full recursion does follow automatically from iteration, according to Prop. 1.2.

Thus, we organize the target set $I\!N$ as a Σ_S-FR-model $A = (A, \boxed{\mathsf{Emp}}^A, (\boxed{\mathsf{Insert}_i}^A)_{i \in \mathbf{item}}, (\boxed{\mathsf{fresh}_i}^A)_{i \in \mathbf{item}})$ as follows:

- $A = I\!N$;
- $\boxed{\mathsf{Emp}}^A = 0$;
- $\boxed{\mathsf{Insert}_i}^A\ s\ n = \begin{cases} n+1, & \text{if } i \notin s, \\ n, & \text{otherwise;} \end{cases}$
- $\boxed{\mathsf{fresh}_i}^A\ s\ n \iff i \notin s$.

It is immediate to check that A satisfies the clauses from $\mathcal{H}_S$ – the resulted unique FR-morphism prescribed by Prop. 1.2 is therefore a map $\mathsf{card} : \mathbf{P}_f(\mathbf{item}) \to I\!N$ satisfying the following properties, obtained by replacing in the FR-morphism conditions (of commuting with the operations and preserving the relations) the actual definitions of the operations and relations on A:

- $\mathsf{card}\ \emptyset = 0$;
- $\mathsf{card}\ (s \cup \{i\}) = \begin{cases} \mathsf{card}\ s + 1, & \text{if } i \notin s, \\ \mathsf{card}\ s, & \text{otherwise;} \end{cases}$

which are (after we delete the vacuous information on one of the cases), precisely the desired clauses.

We have started with a concrete problem – the definition of the cardinal map on finite sets – whose analysis led to a recursion principle. The principle itself is of a general nature, transcending the particular problem. Indeed, defining a map on finite sets using the empty set and insertion of a fresh element as the constructors is very common practice. The iteration principle helps this common practice by indicating *in advance* what needs to be proved to make the definition go through. This avoids having to rely on ad-hoc "dynamic" combinations of definition and proof that are typically employed in such definitions.

2.5 Terms with bindings as an ordinary data type

By "ordinary data type" we mean "data type specifiable as the initial model of a (classical, first-order) Horn theory", as is the finite set example from the previous section. So, are terms ordinary? When answering this, we should not lose sight of the very purpose of the Horn approach: enabling not *any* iteration/recursion principe, but a *useful* one. In particular, for now we care about a principle that would provide solutions to Problems I and II.

2.5.1 Recursion for terms with bindings and substitution

Having the simpler case warm-up and the guidelines from the previous subsection, we proceed directly with building a solution. The signature should contain the term constructs and freshness and substitution operators, as they are directly involved in the desired clauses in both problems.

We thus define the signature Σ to consist of the following:

- For each variable x, a constant symbol, $\boxed{\mathsf{Var}_x}$;
- A binary operation symbol, $\boxed{\mathsf{App}}$;
- For each variable x, a unary operation symbol, $\boxed{\mathsf{Lm}_x}$;
- For each variable x, a binary operation symbol, $\boxed{\mathsf{subst}_x}$;
- For each variable x, a binary relation symbol, $\boxed{\mathsf{fresh}_x}$.

term becomes a Σ-model by interpreting:
- each $\boxed{\mathsf{Var}_x}$ as x,
- $\boxed{\mathsf{App}}$ as App,
- each $\boxed{\mathsf{Lm}_x}$ as $\mathsf{Lm}\ x$,
- each $\boxed{\mathsf{subst}_x}$ as $\_[\_/x]$,
- each $\boxed{\mathsf{fresh}_x}$ as $\mathsf{fresh}\ x$.

We define the Horn theory $\mathcal{H}$ over the signature Σ to consist of the following:
- (1) Definition-like clauses for freshness (w.r.t. the syntactic constructs):

F1: $\boxed{\mathsf{fresh}_z}\ \boxed{\mathsf{Var}_x}$,
for each $x, z \in \mathbf{var}$ with $x \neq z$;[7]

F2: $\boxed{\mathsf{fresh}_z}\ u\ \&\ \boxed{\mathsf{fresh}_z}\ v \Longrightarrow \boxed{\mathsf{fresh}_z}\ (\boxed{\mathsf{App}}\ u\ v)$,
for each $z \in \mathbf{var}$;[8]

[7] That is to say: we take one such Horn clause for each combination of variables x and z (here, such that $x \neq z$); and similarly below.

[8] As usual when one lists Horn clauses (or, in general, FO formulas), the involved FO-variables, here u and v, are assumed to be fixed and distinct.

F3: $\boxed{\mathsf{fresh}_z}\ (\boxed{\mathsf{Lm}_z}\ u)$,
for each $z \in \mathbf{var}$;

F4: $\boxed{\mathsf{fresh}_z}\ u \Longrightarrow \boxed{\mathsf{fresh}_z}\ (\boxed{\mathsf{Lm}_x}\ u)$,
for each $x, z \in \mathbf{var}$;

-(2) Partial definition-like clauses for substitution (again, w.r.t. the syntactic constructs):[9]

S1: $\boxed{\mathsf{subst}_z}\ \boxed{\mathsf{Var}_z}\ w \equiv w$,
for each $z \in \mathbf{var}$;

S2: $\boxed{\mathsf{subst}_z}\ \boxed{\mathsf{Var}_x}\ w \equiv \boxed{\mathsf{Var}_x}$,
for each $x, z \in \mathbf{var}$ with $x \neq z$;

S3: $\boxed{\mathsf{subst}_z}\ (\boxed{\mathsf{App}}\ u\ v)\ w \equiv \boxed{\mathsf{App}}\ (\boxed{\mathsf{subst}_z}\ u\ w)\ (\boxed{\mathsf{subst}_z}\ v\ w)$,
for each $z \in \mathbf{var}$;

S4: $\boxed{\mathsf{fresh}_x}\ w \Longrightarrow \boxed{\mathsf{subst}_z}\ (\boxed{\mathsf{Lm}_x}\ u)\ w \equiv \boxed{\mathsf{Lm}_x}\ (\boxed{\mathsf{subst}_z}\ u\ w)$,
for each $x, z \in \mathbf{var}$ with $x \neq z$;

-(3) A clause, "Abstraction Renaming", for renaming variables in abstractions:

AR: $\boxed{\mathsf{fresh}_y}\ u \Longrightarrow \boxed{\mathsf{Lm}_y}\ (\boxed{\mathsf{subst}_x}\ u\ \boxed{\mathsf{Var}_y}) \equiv \boxed{\mathsf{Lm}_x}\ u$,
for each $x, y \in \mathbf{var}$ with $x \neq y$.

The fact that **term** satisfies the above clauses is well-known. For instance, the satisfaction of S4 means:

$$\forall x, z \in \mathbf{var},\ X, Z \in \mathbf{term}.\ x \neq z \wedge \mathsf{fresh}\ x\ Z \implies (\mathsf{Lm}\ x\ X)[Z/z] = \mathsf{Lm}\ x\ (X[Z/z])$$

which is the well-known property of substitution commuting with Lm under appropriate freshness assumptions.

The choice of these clauses was based on the following technical goals, which in turn are supposed to ensure that **term** is the initial model:
- Goal 1: Freshness and substitution need to be "reduced", by definition-like clauses, to the syntactic constructs.
- Goal 2: At the same time, α-equivalence needs to be enforced on (the terms built using) the syntactic constructs.
- Goal 3: The above need to be achieved economically, i.e, not adding any auxiliary operators and using as few and as weak clauses as possible.

Goals (1) and (2) are seen to be mandatory, if one recalls the standard construction of the initial model of a Horn theory, by quotienting the absolutely free model (here, the model

[9] We call them "partial" because, in the Lm-case, they only cover the subcase where the bound variable is conveniently fresh.

of quasi-terms). Goal (3) has a practical purpose: a convenient recursion principle should employ a weak Horn theory, as the potential user of the recursion principle would have to check its clauses to have the definition go through. The desire not to employ additional operators led us to not attempt reducing completely (by Horn clauses) substitution in the Lm-case; fortunately however, complete reduction of substitution does take place *up to the emerging notion of equality*, which turns out to coincide with α-equivalence. The novelty of our reduction technique thus consists in its loose character. In a way, the above Horn theory defines substitution *together* with equality, hence allows itself indulging in some "laziness" when reducing substitution, unlike in the standard approach. (The standard approach is recalled in Section 1.4.1; since substitution is "eagerly" defined/reduced there, an arbitrary pickFresh auxiliary operator is required.)

The above Horn theory does achieve the desired initiality:

Theorem 2.9 *The model* **term** *is the initial* $(\Sigma, \mathcal{H})$*-model, i.e., is (isomorphic to)* $I_{(\Sigma,\mathcal{H})}$.

Before sketching a proof of this theorem, let us spell it out more nicely, doing away with the Horn boxed symbols. We define the following fresh-substitution models, which are models for the signature Σ under a more convenient notation. Because they have operations matching those for terms and are meant to satisfy clauses also satisfied by terms, we think of the fresh-substitution models as being inhabited by term-like items, that we call *generalized terms*, ranged by gX, gY, gZ. Their operators shall also be called *generalized operators*. (We annotate all the names of generalized items by adding "g" as a prefix or as a superscript.) For a fixed fresh-substitution model A, we shall write:

- gVar x, instead of $\boxed{\mathsf{Var}_x}^A$;
- gApp gX gY, instead of $\boxed{\mathsf{App}}^A$ gX gY;
- gLm x gX, instead of $\boxed{\mathsf{Lm}_x}^A$ gX;
- gFresh x gX, instead of $\boxed{\mathsf{fresh}_x}^A$ gX;
- $gX[gY/y]^g$, instead of $\boxed{\mathsf{subst}_y}^A$ gX gY.

Thus, a *fresh-substitution-model* (*FSb-model* for short) consists of a set A (ranged over by gX, gY, gZ) together with operations and relation

- gVar : $\mathbf{var} \to A$,
- gApp : $A \to A \to A$,
- gLm : $\mathbf{var} \to A \to A$,
- $\_[\_/\_]^g : A \to A \to \mathbf{var} \to A$,
- gFresh : $\mathbf{var} \to A \to \mathbf{bool}$,

satisfying the following properties (assumed universally quantified over all their parameters):

F1: $x \neq z \Longrightarrow \mathsf{gFresh}\ z\ (\mathsf{gVar}\ x)$,

F2: $\mathsf{gFresh}\ z\ gX\ \wedge\ \mathsf{gFresh}\ z\ gY \Longrightarrow \mathsf{gFresh}\ z\ (\mathsf{gApp}\ gX\ gY)$,

F3: $\mathsf{gFresh}\ z\ (\mathsf{gLm}\ z\ gX)$,

F4: $\mathsf{gFresh}\ z\ gX \Longrightarrow \mathsf{gFresh}\ z\ (\mathsf{gLm}\ x\ gX)$,

S1: $(\mathsf{gVar}\ z)[gZ/z]^g = gZ$,

S2: $x \neq z \Longrightarrow (\mathsf{gVar}\ x)[gZ/z]^g = \mathsf{gVar}\ x$,

S3: $(\mathsf{gApp}\ gX\ gY)[gZ/z]^g = \mathsf{gApp}\ (gX[gZ/z]^g)\ (gY[gZ/z]^g)$,

S4: $x \neq z\ \wedge\ \mathsf{gFresh}\ x\ gZ \Longrightarrow (\mathsf{gLm}\ x\ gX)[gZ/z]^g = \mathsf{gLm}\ x\ (gX[gZ/z]^g)$,

AR: $x \neq y\ \wedge\ \mathsf{gFresh}\ y\ gX \Longrightarrow \mathsf{gLm}\ y\ (gX[(\mathsf{gVar}\ y)/x]^g) = \mathsf{gLm}\ x\ gX$.

Theorem 2.9 (rephrased) *Let A be an FSb-model. Then there exists a unique map $H : \mathbf{term} \to A$ commuting with the constructors, i.e.,*

- $H\ x = \mathsf{gVar}\ x$,
- $H\ (\mathsf{App}\ X\ Y) = \mathsf{gApp}\ (H\ X)\ (H\ Y)$,
- $H\ (\mathsf{Lm}\ x\ X) = \mathsf{gLm}\ x\ (H\ X)$.

Additionally, H commutes with substitution and preserves freshness, i.e.,

- $H\ (Y[X/x]) = (H\ Y)[(H\ X)/x]^g$,
- $\mathsf{fresh}\ x\ X \Longrightarrow \mathsf{gFresh}\ x\ (H\ X)$.

We call the above map H the *FSb-morphism to A*.

The careful reader may have noticed that the above rephrasing also brings a small strengthening of the theorem on the uniqueness part. Indeed, a pure rephrasing would read "Then there exists a unique map $H : \mathbf{term} \to A$ commuting with the constructors *and commuting with substitution and preserving freshness*", while we stated the theorem noticing that commuting with the constructs is enough to ensure uniqueness. Indeed, it is a standard fact in universal algebra that a *sufficiently complete set of operators*, as are Var, App and Lm here for **term**, are enough to ensure uniqueness [62]. Alternatively, once the existence part of Th. 2.9 has been proved, the strengthening follows easily by induction on terms. This

strengthening is however a minor point, since in general one cares about the existence of a recursor/iterator, not about the uniqueness of the conditions that define the recursive map.[10]

Proof sketch of Th. 2.9 (in the rephrased version notation).

I. Existence.

First of all, we should remark that, in the light of the technical goals (1) and (2) above, the desired fact is rather intuitive, and can be proved in many different ways. This being said, it is true that there are many basic facts about α-equivalence of quasi-terms that need to be available for a proof to go through.

The first step in our proof is to remark the following: if $H : \mathbf{term} \to A$ is an $(\mathsf{Var}, \mathsf{App}, \mathsf{Lm})$-morphism, i.e., if it commutes with the indicated operators, then it also commutes with substitution and preserves freshness. This can be proved by induction on terms, using F1-F4, S1-S4, and the fact that **term** itself satisfies the corresponding clauses. This fact allows us to focus on the simpler task of finding a $(\mathsf{Var}, \mathsf{App}, \mathsf{Lm})$-morphism between **term** and A.

Let $F : \mathbf{qTerm} \to A$ be the unique $(\mathsf{Var}, \mathsf{App}, \mathsf{Lm})$-morphism given by the absolute initiality of **qTerm**. I.e., F is defined by standard recursion:

- $F(\mathsf{qVar}\ x) = \mathsf{gVar}\ x$;
- $F(\mathsf{qApp}\ P\ Q) = \mathsf{gApp}\ (F\ P)\ (F\ Q)$;
- $F(\mathsf{qLm}\ x\ P) = \mathsf{gLm}\ x\ (F\ P)$.

Using F1-F4, one can show by induction on quasi-terms that F preserves freshness, namely:

(1) $\mathsf{qFresh}\ x\ P \Longrightarrow \mathsf{gFresh}\ x\ (F\ P)$.

Next, using (1), S1-S4, F1-F4 and AR, one can show by induction on quasi-terms that, under an appropriate freshness assumption, F commutes with swapping versus unary substitution, namely:

(2) $\mathsf{qFresh}\ y_1\ P \Longrightarrow F\ (P[y_1 \wedge y]^q) = (F\ P)[(\mathsf{gVar}\ y_1)/z]^g$.

(Above, considering swapping on the left-hand side of the equality is much more convenient than considering variable-for-variable substitution, as the latter is not well-behaved on quasi-terms.)

Next, using (1), (2), S1-S4, F1-F4 and AR, S1-S4, one can show by induction on quasi-terms that F respects α-equivalence, i.e.,

(3) $P \simeq_\alpha Q$ implies $F\ P = F\ Q$.

Finally, (3), together with the definitional clauses of F and the fact that α is a congruence on quasi-terms imply standardly (by the universal property of quotient FOL models) the existence of a $(\mathsf{Var}, \mathsf{App}, \mathsf{Lm})$-morphism $H : \mathbf{term} \to A$, as desired.

II. Uniqueness. By an easy induction on terms. $\square$

[10] Most theorem provers supporting recursion neglect the uniqueness aspect. So do works on recursion for syntax with bindings such as [148] and [104].

In most of the cases, the iteration principle from the previous theorem is enough. Occasionally, however, one needs to employ full recursion, which we describe next in the "rephrased" version directly:

A *full-recursion-fresh-substitution-model* (*FR-FSb-model* for short) consists of a set A (ranged over by gX, gY, gZ) together with operations and relation

- $\mathsf{gVar} : \mathbf{var} \to A$,
- $\mathsf{gApp} : \mathbf{term} \to A \to \mathbf{term} \to A \to A$,
- $\mathsf{gLm} : \mathbf{var} \to \mathbf{term} \to A \to A$,
- $(\_\,,\_\,)[(\_\,,\_\,)/\_\,]^g : \mathbf{term} \to A \to \mathbf{term} \to A \to \mathbf{var} \to A$,
- $\mathsf{gFresh} : \mathbf{var} \to \mathbf{term} \to A \to \mathbf{bool}$,

satisfying the following properties (assumed universally quantified over all their parameters):

F1: $x \neq z \Longrightarrow \mathsf{gFresh}\ z\ (\mathsf{gVar}\ x)$,

F2: $\mathsf{fresh}\ z\ X\ \wedge\ \mathsf{gFresh}\ z\ gX\ \wedge\ \mathsf{fresh}\ z\ Y\ \wedge\ \mathsf{gFresh}\ z\ gY \Longrightarrow \mathsf{gFresh}\ z\ (\mathsf{gApp}\ X\ gX\ Y\ gY)$,

F3: $\mathsf{gFresh}\ z\ (\mathsf{Lm}\ z\ X)\ (\mathsf{gLm}\ z\ X\ gX)$,

F4: $\mathsf{fresh}\ z\ X\ \wedge\ \mathsf{gFresh}\ z\ X\ gX \Longrightarrow \mathsf{gFresh}\ z\ (\mathsf{Lm}\ x\ gX)\ (\mathsf{gLm}\ x\ X\ gX)$,

S1: $(\mathsf{Var}\ z, \mathsf{gVar}\ z)[(Z, gZ)/z]^g = gZ$,

S2: $x \neq z \Longrightarrow (x, \mathsf{gVar}\ x)[(Z, gZ)/z]^g = \mathsf{gVar}\ x$,

S3: $(\mathsf{App}\ X\ Y,\ \mathsf{gApp}\ X\ gX\ Y\ gY)\ [(Z, gZ)/z]^g =$
$\mathsf{gApp}\ (X[Z/z])\ ((X, gX)[(Z, gZ)/z]^g)\ (Y[Z/z])\ ((Y, gY)[(Z, gZ)/z]^g)$,

S4: $x \neq z\ \wedge\ \mathsf{fresh}\ x\ Z\ \wedge\ \mathsf{gFresh}\ x\ Z\ gZ \Longrightarrow$
$(\mathsf{Lm}\ x\ X,\ \mathsf{gLm}\ x\ X\ gX)\ [(Z, gZ)/z]^g = \mathsf{gLm}\ x\ (X[Z/z])\ ((X, gX)[(gZ, Z)/z]^g)$,

AR: $x \neq y\ \wedge\ \mathsf{fresh}\ y\ X\ \wedge\ \mathsf{gFresh}\ y\ X\ gX \Longrightarrow$
$\mathsf{gLm}\ y\ (X[(\mathsf{Var}\ y)/x])\ ((X, gX)[(\mathsf{Var}\ y, \mathsf{gVar}\ y)/x]^g) = \mathsf{gLm}\ x\ X\ gX$.

Theorem 2.10 *Let A be a FR-FSb-model. Then there exists a unique map $H : \mathbf{term} \to A$ commuting with the constructors, i.e.,*

- $H\ x = \mathsf{gVar}\ x$,
- $H\ (\mathsf{App}\ X\ Y) = \mathsf{gApp}\ X\ (H\ X)\ Y\ (H\ Y)$,

- $H\ (\mathsf{Lm}\ x\ X) = \mathsf{gLm}\ x\ X\ (H\ X)$.

Additionally, H commutes with substitution and preserves freshness, i.e.,

- $H\ (Y[X/x]) = (Y, H\ Y)[(X, H\ X)/x]^g$,
- $\mathsf{fresh}\ x\ X \Longrightarrow \mathsf{gFresh}\ x\ X\ (H\ X)$.

We call the above map H the *FR-FSb-morphism to A*.

Proof sketch. FR-FSb-models are a rephrasing of the notion of Σ-FR-model, just like FR-models are a rephrasing of the notion of Σ-model. Moreover, this theorem is a rephrasing of the existence and uniqueness of a FR-morphism (in Σ).

All these mean that this theorem follows from Th. 2.9 by a general Horn argument, namely, that from the proof of Prop. 1.2. □

2.5.2 Solutions for the two problems

We are finally ready to present the promised solutions. As prescribed by Section 2.5.1, we organize the target sets as models for the discussed Horn theories; then we use Th. 2.9 (rephrased).

For Problem I. We let A, the set of *semantic values*, ranged over by s, t, be $\mathbf{val} \to D$. We organize A as an FSb-model. The desired recursive equations from Section 2.3.1 leave us no choice about the generalized construct operators on A:
- $\mathsf{gVar}\ x = \lambda\rho.\ \rho\ x$;
- $\mathsf{gApp}\ s\ t = \lambda\rho.\ \mathsf{APP}\ (s\ \rho)\ (t\ \rho)$;
- $\mathsf{gLm}\ x\ s = \lambda\rho.\ \mathsf{LM}\ (\lambda d.\ s\ (\rho[x \leftarrow d]))$.

Moreover, the desired freshness obliviousness property imposes that generalized freshness on A be at least as strong as the following relation, and we choose to define it as precisely this relation:
- $\mathsf{gFresh}\ x\ s = (\forall \rho, \rho'.\ \rho =_x \rho' \Longrightarrow s\ \rho = s\ \rho')$.

Finally, the desired substitution compositionality leaves us no choice about the definition of generalized substitution on A:
- $s[t/x]^g = \lambda\rho.\ s\ (\rho[x \leftarrow t\ \rho])$.

Thus, saying that the clauses and desired facts listed in Section 2.3.1 hold is the same as saying that the map $[\_]$ is an FSb-morphism. For instance, substitution compositionality, i.e.,
$\forall X\ Y\ X\ \rho.\ [Y[X/x]]\ \rho = [Y]\ (\rho[x \leftarrow [X]\ \rho])$
means
$\forall X\ Y\ X.\ [Y[X/x]] = \lambda\ \rho.\ [Y]\ (\rho[x \leftarrow [X]\ \rho])$,
which means

$\forall X\ Y\ X.\ [Y[X/x]] = [Y]\,[[X]/x]^g$,
which means: $[\_]$ commutes with substitution.

According to Th. 2.9 (rephrased), what remains to do in order to have Problem I resolved is checking that A with the above structure is indeed an FSb-model, i.e., satisfies the clauses F1-F4, S1-S4 and AR. It turns out that here, *as well as in most of the examples we consider later*, checking these facts is trivial.

For instance, checking the most complex of them, AR, amounts to:
- fixing s, ρ, d, x, y,
- assuming $\forall\ \rho_1\ \rho_2.\ \rho_1 =_y \rho_2 \Longrightarrow s\ \rho_1 = s\ \rho_2$,
- and showing that $s(\rho[y \leftarrow d][x \leftarrow d]) = s(\rho[x \leftarrow d])$.

The latter fact follows immediately from the assumption, noticing that $\rho[y \leftarrow d][x \leftarrow d] =_y \rho[x \leftarrow d]$.

For Problem II. We take A to be $\mathbf{term}(\{\mathsf{APP}, \mathsf{LM}\})$ (the set of λ-terms with two constants). We organize A as an FSb-model.

Again, the generalized construct operators, the minimal notion of freshness, and the generalized substitution on A are all pre-determined by the desired recursive equations:
- $\mathsf{gVar}\ x = x$;
- $\mathsf{gApp}\ X\ Y = \mathsf{App}\ (\mathsf{App}\ \mathsf{APP}\ X)\ Y$;
- $\mathsf{gLm}\ x\ X = \mathsf{App}\ \mathsf{LM}\ (\mathsf{Lm}\ x\ X)$;
- $\mathsf{gFresh} = \mathsf{fresh}$;
- $\_[\_/\_]^g = \_[\_/\_]$.

Notice that gApp and gLm are different from App and Lm, while, to the contrary, gFresh and $\_[\_/\_]^g$ are just regular freshness and substitution – a main point of HOAS is reusing substitution. Again, it is immediate to check that A is an FSb-model, yielding, by Th. 2.9 (rephrased), the existence (and uniqueness) of a map $\mathsf{rep} : \mathbf{term} \to \mathbf{term}(\{\mathsf{APP}, \mathsf{LM}\})$ satisfying the desired recursive equations and in addition preserving freshness and being compositional w.r.t. substitution. (Our Isabelle formalization of this example is presented in detail in Section 2.9.4.)

2.5.3 Bottom line to our solutions

There are two advantages in employing the described recursors instead of directly attempting a representative-independent definition (perhaps proved to be so, dynamically, at definition-time):

- Clear aprioric picture of what needs to be proved.
- Extra built-in compositionality results.

Concerning the first of these, the reader may wonder whether the proofs of the involved Horn clauses are really necessary, and weather the aforementioned direct approach may not in fact be faster. To get an answer to this, the reader should attempt such a direct proof – he/she may find amusing contemplating the *unavoidability* of these (or very similar) clauses, that we claim would have to be (re)discovered, possibly slowly and painfully, within the direct approach.

2.6 More examples

Our Horn-based principle is (perhaps surprisingly) extremely general. We could not find in the literature any example of a syntactic map supposed to feature compositionality (of some kind) w.r.t. substitution and not falling in the scope of this principle. In this section we give more examples.

As seen in the solutions for Problems I and II, listing all the desired clauses – those for the term constructs, the minimal one for freshness, and the one for substitution – determines a model automatically. Therefore, when discussing the next examples, we shall only spell the clauses, leaving implicit the construction of the corresponding model. Unless otherwise specified, the considered syntax is that of the untyped λ-calculus, **term**.

2.6.1 The number of free occurrences of a variable in a term

Given a variable z, the intended definitional clauses of the map $\mathsf{no}^z : \mathbf{term} \to I\!N$, taking any term X to the number of free occurrences of z in X, are the following:

- $\mathsf{no}^z\ x = \mathsf{if}\ x = z\ \mathsf{then}\ 1\ \mathsf{else}\ 0$;
- $\mathsf{no}^z(\mathsf{App}\ X\ Y) = (\mathsf{no}^z\ X) + (\mathsf{no}^z\ Y)$;
- $\mathsf{no}^z(\mathsf{Lm}\ x\ X) = \mathsf{if}\ x = z\ \mathsf{then}\ 0\ \mathsf{else}\ \mathsf{no}^z\ X$.

To make them into a rigorous definition, we need to ask: what is the desired/necessary relationship between no^z and freshness on one hand and no^z and substitution on the other. (In fact, an employment of this map in larger developments would typically need to ask these questions anyway.)

The answer to the the first question is the simpler one:

- $\mathsf{fresh}\ z\ X \implies \mathsf{no}^z\ X = 0$.

(The above happens to hold as an "iff", but this is not needed for our purpose, where a "preservation" clause suffices.)

To the substitution question however, we cannot answer for a fixed z, but need to consider all no^z's at the same time, obtaining:

- $\mathsf{no}^z(X[Y/y]) = \begin{cases} (\mathsf{no}^y\ X) * (\mathsf{no}^z\ Y), & \text{if } y = z \\ (\mathsf{no}^z\ X) + (\mathsf{no}^y\ X) * (\mathsf{no}^z\ Y), & \text{otherwise} \end{cases}$

This of course suggests considering the simultaneous version of the map, $\mathsf{no} : \mathbf{term} \to (\mathbf{var} \to I\!N)$, where $\mathsf{no}\ X\ z$ now denotes what used to be $\mathsf{no}^z\ X$. We obtain:
- $\mathsf{no}\ x\ z =$ if $x = z$ then 1 else 0;
- $\mathsf{no}\,(\mathsf{App}\ X\ Y)\ z = (\mathsf{no}\ X\ z) + (\mathsf{no}\ Y\ z)$;
- $\mathsf{no}\,(\mathsf{Lm}\ x\ X)\ z =$ if $x = z$ then 0 else $\mathsf{no}\ X\ z$.
- $\mathsf{fresh}\ z\ X \implies \mathsf{no}\ X\ z = 0$;
- $\mathsf{no}\,(X[Y/y])\ z = \begin{cases} (\mathsf{no}\ X\ y) * (\mathsf{no}\ Y\ z), & \text{if } y = z \\ (\mathsf{no}\ X\ z) + (\mathsf{no}\ X\ y) * (\mathsf{no}\ Y\ z), & \text{otherwise} \end{cases}$

which do work as a Horn definition of no. (Checking the desired clauses is trivial arithmetic.)

2.6.2 A more technically involved example – connection with the de Bruijn representation

De Bruijn indexes are a standard way to represent λ-calculus for efficient manipulation / implementation purposes. The set **dB**, of *de Bruijn terms*, ranged over by K, L, M, is defined by the following grammar, where n ranges over $I\!N$:

$$K \quad ::= \quad \mathsf{VAR}\ n \mid \mathsf{APP}\ K\ L \mid \mathsf{LM}\ K$$

There are no variable bindings involved – rather, the index n of a de Bruijn variable $\mathsf{VAR}\ n$ indicates the "distance" between the given occurrence and its LM-binder (if any), i.e., the number of LM-operators interposed between that occurrence and its binder. Thus, e.g., the λ-term $\mathsf{Lm}\ x\ (\mathsf{Lm}\ y\ (\mathsf{App}\ y\ x))$ corresponds to the de Bruijn term $\mathsf{LM}\ (\mathsf{LM}\ (\mathsf{APP}\ (\mathsf{VAR}\ 0)\ (\mathsf{VAR}\ 1)))$.

As it stands, this correspondence is not perfect, since, on one hand, there are λ-terms with free variables, and, on the other, there are de Bruijn terms with dangling indexes, i.e., indexes that fail to point to a valid binder (e.g., $\mathsf{VAR}\ 0$, or $\mathsf{LM}\ (\mathsf{VAR}\ 1)$). A more serious discrepancy is that the processes of binding variables in terms are different: using Lm, one can choose any variable x to be bound, while using LM one binds the "default" de Bruijn variables waiting to be bound, namely, those with dangling level 0.

A closer mathematical connection, allowing one to actually relate de Bruijn versions of standard results with the "official" λ-versions (as is done, e.g., in [107]) can be achieved via parameterizing by variable orderings, as discussed in [33].

We model such orderings as injective valuations of variables into numbers, i.e., injective elements of $\mathbf{val} = (\mathbf{var} \to I\!N)$. Let **sem**, the set of semantic values, ranged over by s, be $\{\rho \in \mathbf{val}.\ \mathsf{inj}\ \rho\} \to \mathbf{dB}$, where $\mathsf{inj} : \mathbf{val} \to \mathbf{bool}$ is the predicate stating that a valuation is injective. Then λ-terms are interpreted as de Bruijn terms by a map $\mathsf{toDB} : \mathbf{term} \to \mathbf{sem}$, read "to de Bruijn" as follows.

The clauses for variables and application are the obvious ones:
- (1) $\mathsf{toDB}\ x = \lambda\rho.\ \mathsf{VAR}(\rho\ x)$;

- (2) $\mathsf{toDB}\ X\ Y = \lambda\rho.\ \mathsf{APP}\ (\mathsf{toDB}\ X\ \rho)\ (\mathsf{toDB}\ Y\ \rho)$.

To interpret λ-abstraction, we first define $\mathsf{mkFst} : \mathbf{var} \to \mathbf{val} \to \mathbf{val}$, read "make first":
- $\mathsf{mkFst}\ x\ \rho = \lambda\ y.\ \mathsf{if}\ y = x\ \mathsf{then}\ 0\ \mathsf{else}\ (\rho\ y + 1)$.
(Thus, x is "made first" by assigning it the first position, 0, in ρ, and by making room for it through shifting the values of all the other variables.) Now, we set
- (3) $\mathsf{toDB}\ (\mathsf{Lm}\ x\ X) = \lambda\rho.\ \mathsf{LM}(\mathsf{toDB}\ X\ (\mathsf{mkFst}\ x\ \rho))$.
(Thus, in order to map $(\mathsf{Lm}\ x)$-abstractions to LM-abstractions, x is initially "made first", i.e., mapped to 0, the only value which LM "knows" how to bind.)

The above clauses are essentially Definition 2.3 on page 192 in [33] (under a slightly different notation). There are some similarities between these clauses and those from Problem I, but there are also some major differences, notably the interpretation of λ-abstractions and the restricted space of valuations here, which prevent us from reducing one problem to the other.

As usual, to make the definition rigorous, we ask the freshness and substitution questions.

What can we infer about x versus $\mathsf{toDB}\ X$ if we know that x is fresh for X? The answer to this is similar to that from Problem I: we can infer obliviousness to x of the interpretation:
- (4) $\mathsf{fresh}\ x\ X \Longrightarrow (\forall\ \rho\ \rho'.\ \rho =_x \rho' \Longrightarrow \mathsf{toDB}\ X\ \rho = \mathsf{toDB}\ X\ \rho')$.

As for the substitution question, the answer is given by a lemma stated in loc. cit.:
- (5) $\mathsf{toDB}\ (X[Y/y]) = \lambda\rho.\ (\mathsf{toDB}\ X)\ (\mathsf{mkFst}\ y\ \rho)\ [(\mathsf{toDB}\ Y\ \rho)/0]^b$,
where $\_[\_/\_]^b : \mathbf{dB} \to \mathbf{dB} \to I\!N \to \mathbf{dB}$ is de Bruijn substitution (i.e, plain FO-substitution):
— $(\mathsf{VAR}\ n)[M/m]^b = \mathsf{if}\ n = m\ \mathsf{then}\ M\ \mathsf{else}\ \mathsf{VAR}\ n$,
— $(\mathsf{APP}\ K\ L)[M/m]^b = \mathsf{APP}\ (K[M/m]^b)\ (L[M/m]^b)$,
— $(\mathsf{LM}\ K)[M/m]^b = \mathsf{LM}\ (K[M/m]^b)$.
(Thus, the toDB treatment of substitution is similar to that of λ-abstraction: x is 0-indexed and then 0 is de Bruijn-substituted.)

Again, to have clauses (1)-(3) rigorously justified and clauses (4), (5) proved, one needs to check that the resulted model is an FSb-model. Unlike in the previous examples, here there are some complications, in that some of the FSb-model clauses simply do not hold on the whole space **sem**. Fortunately however, a technical closure condition fixes this problem. Namely, if we take
$\mathbf{sem}' = \{s \in \mathbf{sem}.\ \forall \sigma : I\!N \to I\!N,\ \rho \in \mathbf{val}.\ \mathsf{inj}\ \sigma\ \wedge\ \mathsf{inj}\ \rho \Longrightarrow s\ (\sigma \circ \rho) = G\ \sigma\ (\mathsf{toDB}\ X\ \rho)\}$,
where $G : (I\!N \to I\!N) \to \mathbf{dB} \to \mathbf{dB}$ is a map whose definition we give below, then $\mathbf{sem}'$, with the FSb-structure given by clauses (1)-(5), indeed turns out to be an FSb-model – checking this is relatively easy, but does rely on some de Bruijn arithmetic.

The definition of G (having little importance beyond its technical role in the invariant that helps defining toDB), goes as follows:
- $G\ \sigma\ K = \mathsf{vmapDB}\ (\mathsf{cut}\ \sigma)\ 0\ K$, where
- $\mathsf{vmapDB} : (I\!N \to I\!N \to I\!N) \to I\!N \to \mathbf{dB} \to \mathbf{dB}$, read "variable map", is the natural de

Bruijn analogue of, say, list map (also taking into account, like any "natural" de Bruijn function, the LM-depth, recursively):
— $\mathsf{vmapDB}\ \sigma\ n\ (\mathsf{VAR}\ i) = \mathsf{VAR}\ (\sigma\ n\ i)$,
— $\mathsf{vmapDB}\ \sigma\ n\ (\mathsf{APP}\ K\ L) = \mathsf{APP}\ (\mathsf{vmapDB}\ \sigma\ K)\ (\mathsf{vmapDB}\ \sigma\ L)$;
— $\mathsf{vmapDB}\ \sigma\ (\mathsf{LM}\ K) = \mathsf{LM}\ (\mathsf{vmapDB}\ \sigma\ (n+1)\ K)$;
- and $\mathsf{cut} : (I\!N \to I\!N) \to I\!N \to (I\!N \to I\!N)$ is defined by
$\mathsf{cut}\ \sigma\ n\ i = \ \mathsf{if}\ i < n\ \mathsf{then}\ i\ \mathsf{else}\ \sigma\ (i-n) + n$.

For more details on the technical development which lead to the definition of toDB, see Section 2.9.4 and theory C_deBruijn from the (commented) formal scripts available at [121].

We conjecture that the difficulty we encountered with constructing an FSb-model for this example would be matched by a corresponding difficulty in making the definitions go through in an ad-hoc approach, where one would eventually have to discover a similar invariant (again, compared to our approach, with the disadvantage of having to do this discovery by repeated trial and error in the middle of inductive proofs, not knowing in advance what properties need to hold). Unfortunately, we have no way of testing this conjecture, as ours seems to be the first rigorous (let alone formal) treatment of this construction from the literature. In [33], one employs quasi-terms rather than terms, and α-equivalence is not even defined until after defining the above de Bruijn interpretation, and then its definition is non-standard: α-equivalence is taken to be the very kernel of this interpretation. Moreover, the substitution lemma is stated in op. cit. without a proof. (It would of course be interesting to see a Nominal-Logic take on this example.)

2.6.3 A more subtle example – CPS transformation

Consider the task of defining the call-by-value to call-by-name continuation passing-style (CPS) transformation as discussed in [117] in the context of λ-calculus. The transformation goes recursively by the following clauses on the syntax of terms (where below we also let k range over variables thought as continuations):
- $\mathsf{cps}\ x = \mathsf{Lm}\ k\ (\mathsf{App}\ k\ x)$, for some $k \neq x$;
- $\mathsf{cps}\ (\mathsf{Lm}\ x\ X) = \mathsf{Lm}\ k\ (\mathsf{App}\ k\ (\mathsf{Lm}\ x\ (\mathsf{cps}\ X)))$,
for some fresh variable k;
- $\mathsf{cps}\ (\mathsf{App}\ X\ Y) =$
$\mathsf{Lm}\ k\,(\mathsf{App}\ \ (\mathsf{cps}\ X)$
$\qquad\qquad (\mathsf{Lm}\ x\ (\mathsf{App}\,(\mathsf{cps}\ Y)$
$\qquad\qquad\qquad\qquad (\mathsf{Lm}\ y\ (\mathsf{App}\ (\mathsf{App}\ x\ y)\ k)))))$,
for some fresh distinct variables k, x, y.

Above, the most specific clause is the one for App, which follows the usual sequential interpretation of CPS. Namely, let X' and Y' be $\mathsf{cps}\ X$ and $\mathsf{cps}\ Y$, i.e., the CPS-transformed versions of X and Y. Then, given any continuation k, $\mathsf{cps}\ (\mathsf{App}\ X\ Y)$ is evaluated as follows:

- X' is evaluated and the result is passed, as x, to the continuation starting with "$\mathsf{Lm}\ x$",
- which in turn evaluates Y' and passes the result, as y, to the continuation starting with "$\mathsf{Lm}\ y$",
- which in turn evaluates $\mathsf{App}\ x\ y$ and finally passes the result to the original continuation, k.
Consequently, call-by-value behavior is achieved with call-by-name mechanisms. In fact, it is achieved independently of whether call-by-name or call-by-value are assumed as underlying mechanisms on the host syntax – this is known as "indifference" (see Th. 1 on page 148 in op. cit.).

As before, the problem is making the above definition rigorous. Now, trying to approach this directly using our Horn machinery for the **term** syntax does not work for the following reason: the desired transformation does not commute with arbitrary substitution, but only with substitution of *values* for variables, where a value is a term which is either a variable or a Lm-abstraction. I.e., as shown in op. cit. (Lemma 1 on page 149), $\mathsf{cps}\ (X[Yvl/y]) = (\mathsf{cps}\ X)\ [(\mathsf{cps}\ Yvl)/y]$ does hold for all values Yvl, but not for arbitrary terms. Interestingly, the solution lays here in recognizing the proper granularity of the syntax suggested above. Indeed, the aforementioned restricted substitution compositionality, *as well as all the rest of the call-by-value theory developed in op. cit.*, requires that, under call-by-value, the syntactic categories should emphasize value terms. Namely, we are better off if we work with the following variation of **term**, split in two syntactic categories:
- $\mathbf{term}_{\mathit{fl}}$, of *full terms*, ranged over by X, Y,
- $\mathbf{term}_{\mathit{vl}}$, of *value terms*, (or, simply, *values*), ranged over by Xvl, Yvl,
defined mutually recursively by the following grammar:

$$\begin{array}{lcl} X & ::= & \mathsf{InVl}\ Xvl \mid \mathsf{App}\ X\ Y \\ Xvl & ::= & x \mid \mathsf{Lm}\ x\ X \end{array}$$

Thus, $(\mathbf{term}_{\mathit{fl}}, \mathbf{term}_{\mathit{vl}})$ is **term** with values singled out as a separate category (InVl being the injection of values into full terms). Here, only substitution of values for variables makes sense, "institutionalizing" the following semantic remark from op. cit. on page 135: "free variables should be thought of as ranging over values and not arbitrary terms". In this two-sorted context, the corresponding version of our iteration principle does the job of defining the mutually recursive maps $\mathsf{cps} : \mathbf{term}_{\mathit{fl}} \to \mathbf{term}$ and $\mathsf{cps}_{\mathit{vl}} : \mathbf{term}_{\mathit{vl}} \to \mathbf{term}$, together with the (proved) statements of their freshness and substitution preservation, by the clauses:
- (1) $\mathsf{cps}_{\mathit{vl}}\ x = x$;
- (2) $\mathsf{cps}_{\mathit{vl}}\ (\mathsf{Lm}\ x\ X) = \mathsf{Lm}\ x\ (\mathsf{cps}\ X)$;
- (3) $\mathsf{cps}(\mathsf{InVl}\ Xvl) = \mathsf{Lm}\ k\ (\mathsf{App}\ k\ (\mathsf{cps}_{\mathit{vl}}\ Xvl))$, for some fresh variable k;
- (4) $\mathsf{cps}(\mathsf{App}\ X\ Y) = \langle$same as before$\rangle$

- (5) $\mathsf{fresh}\ y\ X \implies \mathsf{fresh}\ y\ (\mathsf{cps}\ X)$;
- (6) $\mathsf{fresh}\ y\ Xvl \implies \mathsf{fresh}\ y\ (\mathsf{cps}_{vl}\ Xvl)$;

- (7) $\mathsf{cps}_{vl}\ (Xvl\,[Yvl/y]) = (\mathsf{cps}_{vl}\ Xvl)\ [(\mathsf{cps}_{vl}\ Yvl)/y]$;
- (8) $\mathsf{cps}\ (X[Yvl/y]) = (\mathsf{cps}\ X)\ [(\mathsf{cps}\ Yvl)/y]$.

Notice that the originally intended behavior for variables and Lm-abstractions now follows from the above, by inlining cps_{vl}, for InVl-wrapped values:
- $\mathsf{cps}\ (\mathsf{InVl}\ x) = \mathsf{Lm}\ k\ (\mathsf{App}\ k\ x)$, for some $k \neq x$;
- $\mathsf{cps}\ (\mathsf{InVl}\ (\mathsf{Lm}\ x\ X)) = \mathsf{Lm}\ k\ (\mathsf{App}\ k\ (\mathsf{Lm}\ x\ (\mathsf{cps}\ X)))$, for some fresh variable k.

Checking the necessary clauses for the definition to work is again routine, provided several basic properties of freshness and substitution are available. There is of course the question whether it is worth "storing" call-by-value λ-calculus in a different data type from standard λ-calculus, or should one rather define values inside standard λ-calculus and work with these. This is an engineering, not a mathematical question, since the two approaches are of course equivalent. If one opts for a different data type, then the isomorphism between $\mathbf{term}_{fl}$ and **term** is yet another (trivial) application of our iteration principle.

2.6.4 An example employing full recursion – freshness of a constant for a term

Here we work with the syntax of λ-calculus with constants. The predicate $\mathsf{ctFresh} : \mathbf{const} \to \mathbf{term}(\mathbf{const}) \to \mathbf{bool}$, where $\mathsf{ctFresh}\ c\ X$ states that c is fresh for (i.e., does not occur in) X, is given recursively by the clauses:[11]
- (1) $\mathsf{ctFresh}\ c\ x = \mathsf{True}$;
- (2) $\mathsf{ctFresh}\ c\ d = (c \neq d)$;
- (3) $\mathsf{ctFresh}\ c\ (\mathsf{App}\ X\ Y) = (\mathsf{ctFresh}\ c\ X\ \wedge\ \mathsf{ctFresh}\ c\ Y)$;
- (4) $\mathsf{ctFresh}\ c\ (\mathsf{Lm}\ x\ X) = \mathsf{ctFresh}\ c\ X$.

Here, posing our usual freshness and substitution questions brings up two interesting phenomena.

What can we infer about x versus (the truth value of) $\mathsf{ctFresh}\ c\ X$ knowing that x is fresh for X? Absolutely nothing! This is OK, since, in the corresponding FR-FSb-model, we can just set gFresh to be vacuously True, thus implementing the uninformative clause:
- (4) $\mathsf{fresh}\ x\ X \implies \mathsf{True}$.

This vacuousness w.r.t. freshness of course suggests we are using a perhaps too complex machinery for a simple case, which here is true. But it also suggests that the impossibility

[11] As with most predicates, this is a case where an inductive definition would perhaps be more natural than recursion; however, we define it recursively since this is a rare occasion to illustrate full recursion.

to infer anything from freshness is not an impediment in making our style of definitions go through. In fact, this holds in general, as a simple consequence of the extra flexibility offered by relations (compared to operations): Let us call *Sb-model* a structure as in the definition of FSb-models (preceding Th. 2.9 (rephrased)), but without any generalized freshness operator gFresh, and satisfying S1-S3 and [S4 and AR with gFresh removed]. Then a fresh-ignoring version of Th. 2.9 (rephrased) holds for it. The reason is that any Sb-model can be made into an FSb-model by just defining gFresh to be vacuously true. And similarly for FR-FSb-models and Th. 2.10.

Now, how can we express ctFresh c $(X[Y/y])$ in terms of ctFresh c X and ctFresh c Y? (Notice that, unlike freshness, substitution is treated as an operation, with which the morphism needs to commute; therefore, being able to answer the "substitution question" is mandatory for our technique to work – see also Section 2.6.6.) Well, we really cannot, unless we are able to test whether y is fresh for X. This means that, in the clause for substitution, namely

- (5) ctFresh c $(X[Y/y]) = (\mathsf{ctFresh}\ c\ X\ \wedge\ (\mathsf{fresh}\ y\ X\ \vee\ \mathsf{ctFresh}\ c\ Y))$,

we need to rely not only on the recursively computed results ctFresh c X and ctFresh c Y, but also on one of the original values, here, X. In other words, we need full recursion, i.e., Th. 2.10.

This may look a little surprising, since the clauses for the syntactic constructs, (1)-(3), traditionally regarded as "the definition", are purely iterative, only the "extra fact" (5) relaying on a full-recursion-like mechanism. This situation reminds one that what should be properly considered as the *definitional clauses* include the "extra facts" for freshness and substitution. Indeed, (1)-(3) do not make a definition before (4) and (5) are stated (and some properties are proved).

2.6.5 Other examples

These include the CPS transformation in the opposite direction (from call-by-name to call-by-value) from [117] and the translation of the LF syntax into untyped λ-calculus (meant to be subsequently Curry-style typed) employed in the proof of strong normalization for the LF reduction [63, 19]. The reader is invited to try our technique on his/her own examples pertaining to syntax with bindings – again, we believe it is very likely to work provided a substitution lemma of some kind is in sight.

2.6.6 Non-examples

Of course, not everything one wants to define is compositional w.r.t. substitution. Such cases fall out of the scope of our definitional principle. These include, e.g., the depth operator – an immediate symptom showing why our principle cannot handle this example is the fact

that we cannot compute the depth of $X[Y/y]$ based on the depths of X and Y;[12] in other words, we cannot answer the substitution question. So this example is problematic because it is, in some sense, syntactically under-specified, as it "ignores" substitution. An example problematic because of a somewhat opposite reason is the so-called *complete development*, introduced in [145] as a means to simplify the proof of the Church-Rosser theorem (and of other results). This map, $\mathsf{cdev} : \mathbf{term} \to \mathbf{term}$, is given by the clauses:

- (1) $\mathsf{cdev}\ x = x$;
- (2) $\mathsf{cdev}\ (\mathsf{Lm}\ x\ X) = \mathsf{Lm}\ x\ (\mathsf{cdev}\ X)$;
- (3) $\mathsf{cdev}\ (\mathsf{App}\ X\ Y) = \mathsf{App}\ (\mathsf{cdev}\ X)\ (\mathsf{cdev}\ Y)$, if $\mathsf{App}\ X\ Y$ is not a β-redex (i.e., if X does not have the form $\mathsf{Lm}\ y\ Z$);
- (3') $\mathsf{cdev}\ (\mathsf{App}\ (\mathsf{Lm}\ y\ X)\ Y) = (\mathsf{cdev}\ X)[(\mathsf{cdev}\ Y)/y]$.

cdev reduces all the (arbitrarily-nested) β-redexes in a term. It does not commute with substitution, mainly because it is not a "purely syntactic" map, but incorporates some operational semantics. Note however that, even if the codomain consists of terms, our technique does not require that substitution on the domain of terms be mapped to *standard* substitution on the codomain, but rather to some well-behaved notion of "generalized substitution". We therefore could define a non-standard substitution (on standard terms) that will eventually yield preservation of substitution, but this would be unnecessarily complicated *and artificial*. Our principle is not intended to have its users work hard to have their definitions go through; rather, it is aimed at situations where the issues of preservation of freshness and substitution appear naturally as further desired properties. In Section 2.8, we present a work-around that handles the cdev operator by a variation of our approach that renounces the substitution compositionality goal, replacing it with the less ambitious one of swapping compositionality.

2.7 Pushing the Horn approach even further

Our Horn approach defines maps as morphisms on an extension of syntax, in particular infers about the defined maps some extra information: compositionality w.r.t. freshness and substitution. Can we infer even more? The answer is: yes, if we prove more things about the constructed model.

Next we consider criteria for three commonly encountered properties:

- (1) reflection of freshness;
- (2) injectiveness;
- (3) surjectiveness.

As usual, we present these criteria on the syntax of the untyped λ-calculus, **term**.

[12] Not even if we are allowed to use X and Y in the restrictive manner of full recursion, as we did in Section 2.6.4.

(1) An FSb-model is said to be *fresh-reversing* if the following facts (converse to clauses F1-F4 from the definition of FSb-models), hold for it:

F1^r: $\mathsf{gFresh}\ z\ (\mathsf{gVar}\ x) \Longrightarrow x \neq z$,

F2^r: $\mathsf{gFresh}\ z\ (\mathsf{gApp}\ gX\ gY) \Longrightarrow (\mathsf{gFresh}\ z\ gX\ \wedge\ \mathsf{gFresh}\ z\ gY)$,

F3-4^r: $\mathsf{gFresh}\ z\ (\mathsf{gLm}\ x\ gX) \Longrightarrow (z = x\ \vee\ \mathsf{gFresh}\ z\ gX)$.

Theorem 2.11 *If an FSb-model A is fresh-reversing, then the iterative map $H : \mathbf{term} \to A$ from Th. 2.9 (rephrased) also reflects freshness (in addition to preserving it), in that the following holds:*
- $\mathsf{fresh}\ x\ (H\ X) = \mathsf{fresh}\ x\ X$.

Proof sketch. Routine induction on X. $\square$

(2) An FSb-model is said to be *construct-injective* if its construct operators are mutually injective, i.e., if the following hold

- $\mathsf{gVar}\ x$, $\mathsf{gApp}\ X\ Y$ and $\mathsf{gLm}\ z\ Z$ are mutually distinct for all x, y, X, Y, Z;
- gVar, gApp (regarded as an uncurried binary operation) and $\mathsf{gLm}\ x$ (for each x) are injective.

Theorem 2.12 *If an FSb-model A is construct-injective, then the iterative map $H : \mathbf{term} \to A$ from Th. 2.9 (rephrased) is injective.*

Proof sketch. One can show that $\forall Y.\ H\ X = H\ Y \Longrightarrow X = Y$ by induction on X. $\square$

(3) An FSb-model A is said to be *inductive* if the following induction principle holds for it for all $\varphi : A \to \mathbf{bool}$: If the following are true:

- $\forall x.\ \varphi\ (\mathsf{gVar}\ x)$,
- $\forall\, gX\ gY.\ \varphi\ gX\ \wedge\ \varphi\ gY \Longrightarrow \varphi\ (\mathsf{gApp}\ gX\ gY)$,
- $\forall\, x\ gX.\ \varphi\ gX \Longrightarrow \varphi\ (\mathsf{gLm}\ x\ gX)$,

then $\forall gX.\ \varphi\ gX$ is also true.

Theorem 2.13 *If an FSb-model A is inductive, then the iterative map $H : \mathbf{term} \to A$ from Th. 2.9 (rephrased) is surjective.*

Proof sketch. By the inductiveness property, taking φ to be $\lambda\ gX.\ \exists X.\ gX = H\ X$ $\square$

Thus, the above theorems integrate even more facts as "built-in"-s of the recursive definition. Examples include:
- the HOAS representation operator from Section 2.3.2 is freshness-reflective and injective;
- the "number of free occurrences" operator from Section 2.6.1 is freshness-reflecting;
- the CPS transformation operator from Section 2.6.3 is freshness-reflective and injective.

Putting together the above theorems, we obtain the following internal characterization of the term model,[13] up to a (freshness reflecting and preserving and substitution preserving) isomorphism:

Theorem 2.14 *If an FSb-model A is fresh-reversing, construct-injective and inductive, then the iterative map $H : \mathbf{term} \to A$ from Th. 2.9 is an isomorphism of FSb-models (where the notion of isomorphism is the standard notion for first-order models).*

Fresh-reversing, construct-injective, inductive FSb-models can be taken as a clean mathematical incarnation of what we previously called "the Platonic notion of syntax with bindings", i.e., the notion that any correct representation is supposed to capture. The basis for this proposal is straightforward:
- on one hand, the properties involved in this specification should clearly be satisfied by λ-terms;
- on the other, these properties characterize an abstract data type uniquely.

The above internal abstract characterization of **term** is certainly not the first one from the literature – see Section 2.10. However, it appears to be the first one expressed in terms of the essential ingredients only, namely, the syntactic constructs, freshness and substitution, and also the first to be "taken seriously", i.e., formalized and put to use (see Section 2.9.5).

2.8 Variations of the Horn-based recursion principle

Here, we present some variations of the Horn-based recursion principles from Section 2.5, obtained by generalizing the swapping operator, aimed at capturing some of our "non-examples" too.

We have exiled this Horn-based recursion variations towards the end of this chapter for two reasons:
- (1) we are not completely sure of the significant usefulness they add to the already presented principles;
- (2) we did not want to clutter the presentation of the theory with a zoo of variations.

[13] It is *internal*, in that it does not rely on things from "outside" the model itself (such as morphisms and other models), as does, for instance, the characterization as initial Horn model.

We shall only discuss the iterative versions of these variations. The extension from iteration to full recursion can be done standardly, as discussed in Section 1.4.2 and already illustrated by the transition from FSb-models to FR-FSb-models from Section 2.5.1.

2.8.1 Swapping-based variations

Built-in compositionality w.r.t. substitution is one of the main conveniences of our recursion principle. However, as shown in Section 2.6.6, sometimes such compositionality simply does not hold, or does not even makes sense. In such cases, we can seek other means of having the definition go through. Compositionality w.r.t. swapping can be one of these means, as discussed below.

A *fresh-swap-model* (*FSw-model* for short) consists of a set A (ranged over by gX, gY, gZ) together with operations and relation

- $\mathsf{gVar} : \mathbf{var} \to A$,
- $\mathsf{gApp} : A \to A \to A$,
- $\mathsf{gLm} : \mathbf{var} \to A \to A$,
- $\_[\_ \wedge \_]^g : A \to \mathbf{var} \to \mathbf{var} \to A$,
- $\mathsf{gFresh} : \mathbf{var} \to A \to \mathbf{bool}$,

satisfying the following properties (assumed universally quantified over all their parameters):

F1-F4: These are the same clauses as in the definition of FSb-models from Section 2.5.1.

Sw1: $(\mathsf{gVar}\ z)[z_1 \wedge z_2]^g = \mathsf{gVar}(z[z_1 \wedge z_2]^v)$,[14]

Sw2: $(\mathsf{gApp}\ gX\ gY)[z_1 \wedge z_2]^g = \mathsf{gApp}\ (gX[z_1 \wedge z_2]^g)\ (gY[z_1 \wedge z_2]^g)$,

Sw3: $\mathsf{gFresh}\ x\ gZ \Longrightarrow (\mathsf{gLm}\ x\ gX)[z_1 \wedge z_2]^g = \mathsf{gLm}\ (x[z_1 \wedge z_2]^v)\ (gX[z_1 \wedge z_2]^g)$,

CSw: $y \notin \{x, x'\}\ \wedge\ \mathsf{gFresh}\ y\ gX\ \wedge\ \mathsf{gFresh}\ y\ gX'\ \wedge\ gX[y \wedge x]^g = gX'[y \wedge x']^g$
$\Longrightarrow \mathsf{gLm}\ x\ gX = \mathsf{gLm}\ x'\ gX'$.

Above, the name "CSw" of the last clause stands for "Congruence for Swapping".

Theorem 2.15 *Let A be an FSw-model. Then there exists a unique map $H : \mathbf{term} \to A$ commuting with the constructors, i.e.,*

- $H\ x = \mathsf{gVar}\ x$,
- $H\ (\mathsf{App}\ X\ Y) = \mathsf{gApp}\ (H\ X)\ (H\ Y)$,

[14] Recall from Section 1.4.1 that $\_[\_ \wedge \_]^v$ is swapping on variables.

- $H\ (\mathsf{Lm}\ x\ X) = \mathsf{gLm}\ x\ (H\ X)$.

Additionally, H commutes with swapping and preserves freshness, i.e.,

- $H\ (Y[z_1 \wedge z_2]) = (H\ Y)[z_1 \wedge z_2]^g$,
- $\mathsf{fresh}\ x\ X \Longrightarrow \mathsf{gFresh}\ x\ (H\ X)$.

.

We call the above map H the *FSw-morphism to A*.

Proof sketch. Quite similar to the proof of Th. 2.9 (rephrased), except that:
- it employs the swapping-based alternative definition of α-equivalence (rather than the substitution-based one) – see Section 1.4.1;
- it is simpler, since swapping interacts with bindings more straightforwardly than substitution.
□

The complete development operator cdev from Section 2.6.6 is indeed compositional w.r.t. swapping, and clauses (1), (2), (3), (3') from there, together with clauses for freshness and swapping,
- (4) $\mathsf{fresh}\ x\ X \Longrightarrow \mathsf{fresh}\ x\ (\mathsf{cdev}\ X)$,
- (5) $\mathsf{cdev}\ (X[z_1 \wedge z_2]) = (\mathsf{cdev}\ X)[z_1 \wedge z_2]$,

turn out to be form a valid definition (by full recursion) using the aforementioned fresh-swap Horn theory.

Interestingly, yet another swapping-based variation, employing a larger theory, can be recognized by a "Horn reading" of work done by Norrish in [104] – this is detailed in Section 2.10.2.

2.8.2 Other variations

A principle with built-in compositionality w.r.t. (freshness and) both substitution and swapping can be achieved in at least two ways, both including the freshness, substitution and swapping clauses F1-F4, S1-S4 and Sw1-Sw3. Then we can choose to incorporate either the substitution-based abstraction renaming clause AR (from the definition of FSb-models from Section 2.5.1), or CSw, the above congruence-for-swapping clause; in a concrete situation, the user should of course choose the one that is easier to check on the given target domain of the definition – the outcome would be the same, namely, a map commuting with the syntactic constructs, freshness, substitution and swapping.

A moderately interesting question is whether it would be possible to exclude both swapping and substitution, or both freshness and substitution, or both freshness and swapping, from

the recursive clauses (while staying first-order, i.e., not getting into second-order issues such as finiteness of the support, as in Nominal Logic). Indeed, recall that the original goal is really to have the definitional clauses *on the syntactic constructs* go through – w.r.t. this "pure" goal, freshness, substitution and swapping are just subsidiaries. (Although we have argued that considering these subsidiaries, especially the first two, is often meaningful in its own right, as it proves useful lemmas "at definition time".) And indeed, freshness can be trivially eliminated from the target models, both in the swapping and the substitution cases, as briefly discussed in Section 2.6.4 – however, the applicability of the resulted principles does not look very attractive.

2.9 Generalization and formalization

In this section, we first describe a (technical, but rather obvious) general setting for an arbitrary syntax with bindings, and then our Isabelle formalization of this setting. Then we discuss the formalization of the examples presented in Section 2.6, and some larger developments that use these examples. Finally, we show how we have employed the term characterization given in Th. 2.14 to create bridges, on which one may transport formal constructions and results, between different Isabelle formalizations of terms with bindings.

2.9.1 General setting – arbitrary syntax with bindings

So far, we described Horn-based iteration and recursion for the particular syntax of untyped λ-calculus, with the understanding that the results can be straightforwardly generalized to any many-sorted syntax with bindings. In fact, we even implicitly employed variants of these results for λ-calculus with constants (in Section 2.3.2) and for λ-calculus with emphasized values (in Section 2.6.3).

Next we describe what notion of many-sorted syntax with bindings we have in mind, in a manner that is close to our Isabelle formalization we discuss later. But before we do this, let us slightly rephrase the λ-calculus quasi-terms and terms (introduced in Section 1.4.1). We distinguish a new syntactic category, that of *quasi-abstractions*, as the set **qAbs**, ranged over by qA, qB, qC, defined mutually recursively with the set **qTerm** of quasi-terms, ranged over by qX, qY, qZ, as follows:

$$\begin{array}{lcl} qX & ::= & \mathsf{qVar}\ x \mid \mathsf{qApp}\ qX\ qY \mid \mathsf{qLam}\ qA \\ qA & ::= & \mathsf{qAbs}\ x\ qX \end{array}$$

Within $\mathsf{qAbs}\ x\ X$, x is thought as being bound in x. This yields again a standard notion of α-equivalence, this time coming under the form of two relations, one on quasi-terms and one on quasi-abstractions. (Note that the previous λ-abstraction operator $\mathsf{qLm} : \mathbf{var} \to$

qTerm $\to$ **qTerm** can be recovered as the composition of qLam and qAbs: $\mathsf{qLm}\ x\ qX = \mathsf{qLam}\ (\mathsf{qAbs}\ x\ qX)$.) As before, α-equivalence brings up the notions of term (the set **term**, ranged over by X, Y, Z) and abstractions (the set **abs**, ranged over by A, B, C) as α-classes of quasi-terms and quasi-abstractions, which can be briefly specified by the grammar:

$$\begin{array}{lcl} X & ::= & \mathsf{Var}\ x \mid \mathsf{App}\ X\ Y \mid \mathsf{Lam}\ A \\ A & ::= & \mathsf{Abs}\ x\ X \end{array}$$

together with an indication that Abs binds x in X. All the standard operators – freshness, swapping, substitution – come in pairs now: one for terms and one for abstractions (and we use the same notation for both). Mirroring the situation for quasi-items, $\mathsf{Lm} : \mathbf{var} \to \mathbf{term} \to \mathbf{term}$, the previous binding operator, can be recovered as $\mathsf{Lm}\ x\ X = \mathsf{Lam}\ (\mathsf{Abs}\ x\ X)$.

The separation between terms and abstractions is a standard procedure for syntax with bindings, going back at least to Milner's treatment of the π-calculus [88]. It has the advantage of being more flexible thanks to the higher granularity, and in general of being also economical thanks to the possibility of reusing abstractions.

In order to appreciate these features, one should consider syntaxes richer than that of untyped λ-calculus. For example, let us specify the syntax of the Edinburgh LF [63] – while doing this, we try to point out the necessary ingredients required to specify a syntax with bindings, preparing our generalization.

This time, we skip the presentation of the quasi-items, and discuss directly the items obtained after one has factored to α-equivalence.

We first need to know what are the syntactic categories (which we shall call *sorts*): we have *object families*, *type families* and *kinds*. Among these, however, only the first two have associated categories of variables (we do not have kind variables). In other words, we have three sorts of terms, say $\mathsf{o}, \mathsf{t}, \mathsf{k}$, among which o, t are also sorts of variables. We let **var** be a fixed infinite set of variables. We are not going to have multiple sets of variables, but simply inject into terms different copies of **var** as many times as necessary (in this case, twice).

We shall let $\mathbf{term}_s$ denote the set of terms of sort s, namely:
- $\mathbf{term}_{\mathsf{o}}$, ranged over by X, Y, Z, shall be the set of object-family terms;
- $\mathbf{term}_{\mathsf{t}}$, ranged over by tX, tY, tZ, shall be the set of type-family terms;
- $\mathbf{term}_{\mathsf{k}}$, ranged over by kX, kY, kZ, shall be the set of kind terms.

Next, we need to indicate the term constructs, two of which are already given by the above inclusion of variable sorts in term sorts:
- $\mathsf{Var}_{\mathsf{o}} : \mathbf{var} \to \mathbf{term}_{\mathsf{o}}$,
- $\mathsf{Var}_{\mathsf{t}} : \mathbf{var} \to \mathbf{term}_{\mathsf{t}}$.

Let us now establish what categories of abstractions we need, i.e., which sorts of variables are bound in which sorts of terms. We shall allow abstracting a sort of variables in a sort of terms only provided one of the desired operators requires such an abstraction. Anticipating

a bit, it will turn out we only need to bind o-variables in all types of terms. Therefore we have:

- $\mathbf{abs}_{\mathsf{o},\mathsf{o}}$, ranged over by A, B, C,
with the associated constructor $\mathsf{Abs}_{\mathsf{o},\mathsf{o}} : \mathbf{var}_{\mathsf{o}} \to \mathbf{term}_{\mathsf{o}} \to \mathbf{abs}_{\mathsf{o},\mathsf{o}}$;
- $\mathbf{abs}_{\mathsf{o},\mathsf{t}}$, ranged over by tA, tB, tC,
with the associated constructor $\mathsf{Abs}_{\mathsf{o},\mathsf{t}} : \mathbf{var}_{\mathsf{o}} \to \mathbf{term}_{\mathsf{t}} \to \mathbf{abs}_{\mathsf{o},\mathsf{t}}$;
- $\mathbf{abs}_{\mathsf{o},\mathsf{k}}$, ranged over by kA, kB, kC,
with the associated constructor $\mathsf{Abs}_{\mathsf{o},\mathsf{k}} : \mathbf{var}_{\mathsf{o}} \to \mathbf{term}_{\mathsf{k}} \to \mathbf{abs}_{\mathsf{o},\mathsf{k}}$.

This concludes the issue of binding: all the above Abs-constructors bind in the same way and are assumed to act, as before, modulo α. This separation of concerns is one of the advantages of employing the intermediate notion of abstraction.

The remaining term constructs map tuples of terms or abstractions to terms and do not introduce any bindings, hence are injective (free) constructs:

- $\mathsf{App} : \mathbf{term}_{\mathsf{o}} \to \mathbf{term}_{\mathsf{o}} \to \mathbf{term}_{o}$ (object application),
- $\mathsf{Lam} : \mathbf{term}_{\mathsf{t}} \to \mathbf{abs}_{\mathsf{o},\mathsf{o}} \to \mathbf{term}_{o}$ (object lambda),[15]
- $\mathsf{Tapp} : \mathbf{term}_{\mathsf{t}} \to \mathbf{term}_{\mathsf{o}} \to \mathbf{term}_{\mathsf{t}}$ (type application),
- $\mathsf{Tlam} : \mathbf{term}_{\mathsf{t}} \to \mathbf{abs}_{\mathsf{o},\mathsf{t}} \to \mathbf{term}_{\mathsf{t}}$ (type lambda),
- $\mathsf{Tprod} : \mathbf{term}_{\mathsf{t}} \to \mathbf{abs}_{\mathsf{o},\mathsf{t}} \to \mathbf{term}_{\mathsf{t}}$ (type product),
- $\mathsf{Type} : \mathbf{term}_{\mathsf{k}}$ (the "type" kind),
- $\mathsf{Kprod} : \mathbf{term}_{\mathsf{t}} \to \mathbf{abs}_{\mathsf{o},\mathsf{k}} \to \mathbf{term}_{\mathsf{k}}$ (kind product).

This concludes the specification of the LF syntax. Compactly, we can write:

$$
\begin{array}{lcl}
A & ::= & \mathsf{Abs}_{\mathsf{o},\mathsf{o}}\ x\ X \\
tA & ::= & \mathsf{Abs}_{\mathsf{o},\mathsf{t}}\ x\ tX \\
kA & ::= & \mathsf{Abs}_{\mathsf{o},\mathsf{k}}\ x\ kX \\
X & ::= & \mathsf{Var}_{o}\ x \mid \mathsf{App}\ X\ Y \mid \mathsf{Lam}\ tX\ A \\
tX & ::= & \mathsf{Var}_{\mathsf{t}}\ x \mid \mathsf{Tapp}\ tX\ Y \mid \mathsf{Tlam}\ tX\ tA \mid \mathsf{Tprod}\ tX\ tA \\
kX & ::= & \mathsf{Type} \mid \mathsf{Kprod}\ tX\ kA
\end{array}
$$

with the understanding that the abstraction constructors are binders.

Summing up, we needed to indicate:

- the sorts of terms, and among these the sorts of variables (or, alternatively, the sorts of variables, the sorts of terms, and an injection between them);
- the rest of the syntactic constructs, taking tuples of terms and abstractions into terms (from the arities of which we can find out what are the needed abstractions and associate corresponding constructors for them).

[15] E.g., in standard mixfix notation for LF, $\mathsf{Lam}\ tX\ (\mathsf{Abs}\ x\ Y)$ would be written $\lambda x : tX.\ Y$.

We now proceed with the generalization. As before, we fix an infinite set of variables, **var**, ranged over by x, y, z. Given any two sets I and A, we let $\mathbf{Input}(I, A)$ be the set of partial functions from I to A, which we call *I-indexed A-inputs*; elements of $\mathbf{Input}(I, \mathbf{term})$ (for some sufficiently large set I) will be used as inputs (i.e., families of arguments) to the operations of the binding signature. Given $f \in \mathbf{Input}(I, A)$ and $g \in \mathbf{Input}(I, B)$, we write $\mathsf{sameDom}\ f\ g$, read "f and g have the same domain", for $\forall\, i \in I.\ (f\ i \text{ defined }) \iff (g\ i \text{ defined })$.

A *binding signature* Σ is a tuple $(\mathbf{index}, \mathbf{bindex}, \mathbf{varSort}, \mathbf{sort}, \mathbf{opSym}, \mathsf{asSort}, \mathsf{arOf}, \mathsf{barOf})$, where:

- **index**, ranged over by i, j, is the set of *indexes* (meant to be used for building families of free arguments for the operators);
- **bindex**, also ranged over by i, j, is the set of *binding indexes* (*bindexes* for short) (meant to be used for building families of bound arguments for the operators);
- **varSort**, ranged over by xs, is the set of *variable sorts* (*varsorts* for short) (representing the various syntactic categories of variables);
- **sort**, ranged over by s, is the set of *sorts* (representing the various syntactic categories of terms);
- **opSym**, ranged over by δ, is the set of *operation symbols*;
- $\mathsf{asSort} : \mathbf{varSort} \to \mathbf{sort}$ is an injective map (this is the aforementioned inclusion/injection of varsorts "as sorts");
- $\mathsf{stOf} : \mathbf{opSym} \to \mathbf{sort}$, read "the (result) sort of";
- $\mathsf{arOf} : \mathbf{opSym} \to \mathbf{Input}(\mathbf{index}, \mathbf{sort})$, read "the (free) arity of";
- $\mathsf{barOf} : \mathbf{opSym} \to \mathbf{Input}(\mathbf{bindex}, \mathbf{varSort} \times \mathbf{sort})$, read "the bound arity of" ("barity of", for short).

We fix a signature Σ as above. We define the predicate $\mathsf{isInBar} : \mathbf{varSort} \times \mathbf{sort} \to \mathbf{bool}$, read "is in a bound arity", by $\mathsf{isInBar}(xs, s) \iff (\exists \delta\ i.\ \mathsf{barOf}\ \delta\ i = (xs, s))$. Thus, isInBar singles out the useful abstractions, the only ones we shall build.

The following sets:

- for each $s \in \mathbf{sort}$, the set $\mathbf{qTerm}(\Sigma, s)$, of *$\Sigma$-quasi-terms of sort s*, ranged over by qX, qY, qZ,
- for each $xs \in \mathbf{varSort}$ and $s \in \mathbf{sort}$ such that $\mathsf{isInBar}(xs, s)$, the set $\mathbf{qAbs}(\Sigma, (xs, s))$, of *$\Sigma$-quasi-abstractions of type* (xs, s), ranged over by qA, qB, qC,
- for each $\delta \in \mathbf{opSym}$, the set $\mathbf{qinp}(\Sigma, \delta)$, of *(well-sorted) Σ-quasi-inputs for δ*, ranged over by *qinp*,
- for each $\delta \in \mathbf{opSym}$, the set $\mathbf{qbinp}(\Sigma, \delta)$, of *(well-sorted) Σ-quasi-[bound inputs]* (*Σ-quasi-binputs* for short) for δ, ranged over by *qbinp*,

are defined mutually recursively by the following clauses:

- (1) $\mathsf{qVar}\ xs\ x \in \mathbf{term}(\Sigma, \mathsf{asSort}\ xs)$;
- (2) if $qinp \in \mathbf{qinp}(\Sigma, \delta)$ and $qbinp \in \mathbf{qbinp}(\Sigma, \delta)$, then $\mathsf{qOp}\ \delta\ qinp\ qbinp \in \mathbf{qTerm}(\Sigma, \mathsf{stOf}\ \delta)$;

- (3) if $\mathsf{isInBar}(xs, s)$ and $qX \in \mathbf{term}(\Sigma, s)$, then $\mathsf{qAbs}\ xs\ x\ qX \in \mathbf{qAbs}(\Sigma, (xs, s))$;
- (4) if $\mathsf{sameDom}\ (\mathsf{arOf}\ \delta)\ qinp$ and $\forall\ i.\ qinp\ i$ defined $\Longrightarrow qinp\ i \in \mathbf{qTerm}(\Sigma, \mathsf{arOf}\ \delta\ i)$, then $qinp \in \mathbf{qinp}(\Sigma, \delta)$;
- (5) if $\mathsf{sameDom}\ (\mathsf{barOf}\ \delta)\ qbinp$ and $\forall\ i.\ qbinp\ i$ defined $\Longrightarrow qbinp\ i \in \mathsf{qAbs}(\Sigma, \mathsf{barOf}\ \delta\ i)$, then $qbinp \in \mathbf{qbinp}(\Sigma, \delta)$;

Here are some explanations:
- Clauses (1) and (2) are for constructing terms:
— (1) by injecting variables x of various varsorts xs, using a constructor that we called "qVar";
— (2) by applying operations from the signature to quasi-inputs and quasi-binputs, using a generic constructor that we called "qOp", taking an operation symbol as first argument; thus, $\mathsf{qOp}\ \delta$ is the operation on quasi-terms corresponding to the operation symbol δ; this operation is applied to a free input (i.e., a family of quasi-terms) and to a bound input (i.e., a family of quasi-abstractions) according to the (free) arity and bound arity of δ, obtaining a quasi-term having as sort the result sort of δ;
- Clause (3) is for building quasi-abstractions from variables x of various varsorts xs and quasi-terms, using a constructor that we called "qAbs";
- Clause (4) is for building quasi-inputs as families of quasi-terms.
- Clause (5) is for building quasi-binputs as families of quasi-abstractions.

All the above constructions are performed with respecting the varsort-to-sort injections, and the sorts, arities and barities of operation symbols as prescribed by the signature Σ. Thus, e.g., clause (4) can be spelled in "mathematical English" as follows: for a quasi-input *qinp* to be well-sorted w.r.t. an operation symbol δ, *qinp* should be defined precisely where the arity of δ is defined, and, if defined on an index i, the corresponding term should have the corresponding sort.

α-equivalence is defined again standardly considering the fact that, in a quasi-abstraction $\mathsf{qAbs}\ xs\ x\ X$, (xs, x), i.e., the variable x thought of as being of varsort xs, is bound in X. Factoring to α-equivalence, we obtain the following sets:
- for each $s \in \mathbf{sort}$, the set $\mathbf{term}(\Sigma, s)$, of Σ-*terms of sort* s, ranged over by X, Y, Z,
- for each $xs \in \mathbf{varSort}$ and $s \in \mathbf{sort}$ such that $\mathsf{isInBar}(xs, s)$, the set $\mathbf{abs}(\Sigma, (xs, s))$, of Σ-*abstractions of sort* (xs, s), ranged over by A, B, C,
- for each $\delta \in \mathbf{opSym}$, the set $\mathbf{inp}(\Sigma, \delta)$, of *(well-sorted)* Σ-*inputs for* δ, ranged over by *inp*,
- for each $\delta \in \mathbf{opSym}$, the set $\mathbf{binp}(\Sigma, \delta)$, of *(well-sorted)* Σ-*[bound inputs]* (Σ-*binputs* for short) for δ, ranged over by *binp*.

We can state the definition of terms and abstractions compactly as follows, where
- we use superscripts s and (xs, s) to indicate membership to $\mathbf{term}(\Sigma, s)$ and $\mathbf{abs}(\Sigma, (xs, s))$,
- we write inputs and binputs as indexed families,
- we inline the definitions of well-sorted inputs and binputs,

- and we state side-conditions in square brackets:

$$\begin{array}{lcl} A^{(xs,s)} & ::= & \mathsf{Abs}\ xs\ x\ X^{s} \quad [\mathsf{isInBar}(xs,s)] \\ X^{\mathrm{asSort}\ xs} & ::= & \mathsf{Var}\ xs\ x \\ X^{\mathrm{stOf}\ \delta} & ::= & \mathsf{Op}\ \delta\ \begin{array}{l}(\text{if } \mathsf{arOf}\ \delta\ i \text{ defined then some } X^{\mathrm{arOf}\ \delta\ i} \text{ else undefined})_{i\in \mathrm{index}} \\ (\text{if } \mathsf{barOf}\ \delta\ i \text{ defined then some } A^{\mathrm{barOf}\ \delta\ i} \text{ else undefined})_{i\in \mathrm{bindex}}\end{array} \end{array}$$

again, with the understanding that the abstraction constructor Abs is a binder.

The freshness, swapping and substitution operators are all standard (defined similarly to those from the particular case of λ-terms from Section 1.4.1). We have:
- for each $ys \in \mathbf{varSort}$
— and each $s \in \mathbf{sort}$, a freshness operator for ys-variables in s-terms,
$\mathsf{fresh}_{ys,s} : \mathbf{var} \to \mathbf{term}(\Sigma, s) \to \mathbf{bool}$;
— and each $(xs, s) \in \mathbf{varSort} \times \mathbf{sort}$ with $\mathsf{isInBar}(xs, s)$, a freshness operator for ys-variables in (xs, s)-abstractions,
$\mathsf{fresh}_{ys,(xs,s)} : \mathbf{var} \to \mathbf{abs}(\Sigma, (xs, s)) \to \mathbf{bool}$;
- for each $ys \in \mathbf{varSort}$
— and each $s \in \mathbf{sort}$, a swapping operator for ys-variables in s-terms,
$\_[\_ \wedge \_]_{ys,s} : \mathbf{var} \to \mathbf{var} \to \mathbf{term}(\Sigma, s) \to \mathbf{term}(\Sigma, s)$;
— and each $(xs, s) \in \mathbf{varSort} \times \mathbf{sort}$ with $\mathsf{isInBar}(xs, s)$, a swapping operator for ys-variables in (xs, s)-abstractions,
$\_[\_ \wedge \_]_{ys,(xs,s)} : \mathbf{var} \to \mathbf{var} \to \mathbf{abs}(\Sigma, (xs, s)) \to \mathbf{abs}(\Sigma, (xs, s))$;
- for each $ys \in \mathbf{varSort}$
— and each $s \in \mathbf{sort}$, a substitution operator for ys-variables (with $(\mathsf{asSort}\ ys)$-terms) in s-terms,
$\_[\_/\_]_{ys,s} : \mathbf{term}(\Sigma, \mathsf{asSort}\ ys) \to \mathbf{var} \to \mathbf{term}(\Sigma, s) \to \mathbf{term}(\Sigma, s)$;
— and each $(xs, s) \in \mathbf{varSort} \times \mathbf{sort}$ with $\mathsf{isInBar}(xs, s)$, a substitution operator for ys-variables (with $(\mathsf{asSort}\ ys)$-terms) in (xs, s)-abstractions,
$\_[\_/\_]_{ys,(xs,s)} : \mathbf{term}(\Sigma, \mathsf{asSort}\ ys) \to \mathbf{var} \to \mathbf{abs}(\Sigma, (xs, s)) \to \mathbf{abs}(\Sigma, (xs, s))$.

In addition, it is often useful to consider *parallel substitution*, in environments. Given the sets A and B, let $A \rightharpoonup B$ be the set of partial functions between A and B having finite domain. The set $\mathbf{env}$, of *environments*, consists of families $(\rho_{xs} : \mathbf{var} \rightharpoonup \mathbf{term}(\Sigma, \mathsf{asSort}\ xs))_{xs \in \mathbf{var}}$, i.e., families of finite-domain partial functions mapping, for each varsort xs, variables (thought of as xs-variables) to terms of the corresponding sort, $\mathsf{asSort}\ xs$. The intention is that, for each xs, the xs-variable x (appearing as $\mathsf{Var}\ xs\ x$ in a given term or abstraction) be substituted in a capture-free manner by $\rho_{xs}\ x$ if the latter is defined. This is achieved,
- for each $s \in \mathbf{sort}$, by the s-parallel substitution operator,
$\_[\_] : \mathbf{term}(\Sigma, s) \to \mathbf{env} \to \mathbf{term}(\Sigma, s)$,
- and for each $(xs, s) \in \mathbf{varSort} \times \mathbf{sort}$ with $\mathsf{isInBar}(xs, s)$, by the (xs, s)-parallel substitution

operator,
$\_[\_] : \mathbf{abs}(\Sigma, (xs, s)) \to \mathbf{env} \to \mathbf{abs}(\Sigma, (xs, s))$.
The above finite-domain assumption is meant to integrate environments (hence parallel substitution) into the fresh-induction mechanisms from Section 2.2.

Parallel substitution is heavily used in the adequacy proofs from our "HOAS on top of FOAS" development presented in Chapter 3.

Note that our notion of binding signature is not as general as one could reasonably imagine. Restrictions include the following:
- (1) There is no "native" concurrent binding of multiple variables. Thus, a binding construct such as, e.g., $\mathsf{case}\ Z\ \mathsf{of}\ (u : U, y : Y) \Rightarrow X$ (the destructor for dependent sums), meant to bind u in $[Y$ and $X]$ and y in X, would need to be modeled here employing two operators, say Δ_1 and Δ_2, as something like: $\Delta_1\ U\ (\mathsf{Abs}\ u\ (\Delta_2\ Y\ (\mathsf{Abs}\ y\ X)))$. This is adequate, but perhaps not as direct as one may wish.
- (2) There are no variable sorts without corresponding term sorts. Consequently, if one wants such "independent" categories of variables, as, e.g., the channel names in the π-calculus [88], one has to embed them into terms having no other constructors. (Note however that embedding variables into terms may pay off in the end, if one eventually decides to add some constructs on the given syntactic category, as in the polyadic π-calculus [138]).

On the other hand, our setting is more general than one would typically expect, in an unconventional direction: since inputs to term constructs are [**index** and **bindex**]-families and **index** and **bindex** are not required to be finite, we can model *infinitary syntax* (featuring infinitarily-branching terms), which may prove convenient in situations where one wishes to import bits and pieces of simple semantics into the syntax, so that one can focus on more complex features. For example, a standard presentation of CCS [87] employs potentially infinite indexed summations $\sum_{i \in I} P_i$ *as part of the syntax* of process terms – while it is true that most of the useful examples could be captured within an alternative finitary syntax by index variables x and a binding operator $\sum\ x : I.\ P(x)$, there is little point in introducing scoping and variable-capture mechanisms for this simple phenomenon of indexing; instead, the previous "semantic scoping" allowed into syntax seems more natural. Likewise, if we have a concurrent programming language whose processes may pass around numbers, the input of a number may be syntactically a binding $\mathsf{Lm}\ x : I\!N.\ P(x)$, or alternatively it may be the application of an infinitary operator $\mathsf{Inp}\ (P_n)_{n \in I\!N}$.

(One may ask whether other notions of binding can be replaced by infinitary operations as above. The answer to this happens to be relevant to the topic of this dissertation. Above, the main point of writing $\mathsf{Inp}\ (P_n)_{n \in I\!N}$ instead of $\mathsf{Lm}\ x : I\!N.\ P(x)$ was to anticipate all possible substitutions for the bound variable, here, obtaining all possible outcomes P_n of replacing x with a number n in $P(x)$. For a less trivial case, such as λ-calculus, this means replacing abstractions $\mathsf{Abs}\ y\ X$ with infinitary families $(X[Y/y])_{Y \in \mathbf{term}}$ – but this is HOAS!)

This concludes our presentation of the general setting. Generalizations of our Horn-based iteration and recursion theorems that we stated for the untyped λ-calculus, namely Theorems 2.9, 2.10, 2.11, 2.12, 2.13 and 2.14, are conceptually straightforward, but notationally quite tedious, without bringing any new insight. We therefore omit stating them here. (But in Section 2.9.3 we point to the precise locations in our Isabelle scripts where we state and prove these theorems, in a fairly readable format.)

2.9.2 Instantiating the general setting to particular cases

For the terms of the untyped λ-calculus. We take the following binding signature Σ:
- since we have only one syntactic category of variables and one of terms,
— **varSort** is a singleton set, say $\{\mathsf{vlm}\}$;
— **sort** is a singleton set, say $\{\mathsf{lm}\}$;
— $\mathsf{asSort} : \mathbf{varSort} \to \mathbf{sort}$ maps vlm to lm;
- **opSym** is taken to contain symbols for application and λ-abstraction, namely, $\mathbf{opSym} = \{\mathsf{app}, \mathsf{lam}\}$;
- **index** is taken to contain slots for each sort in the arity of each operation symbol, here $\mathbf{index} = \{\mathsf{lapp}_1, \mathsf{lapp}_2\}$;
- **bindex** is taken to contain slots for each pair varsort-sort in the barity of each operation symbol, here $\mathbf{bindex} = \{\mathsf{llam}\}$;
- $\mathsf{stOf} : \mathbf{opSym} \to \mathbf{sort}$ is defined by:
— $\mathsf{stOf}\ \mathsf{app} = \mathsf{lm}$;
— $\mathsf{stOf}\ \mathsf{lam} = \mathsf{lm}$;
- $\mathsf{arOf} : \mathbf{opSym} \to \mathbf{Input}(\mathbf{index}, \mathbf{sort})$ is defined by:
— $\mathsf{arOf}\ \mathsf{app}\ i = \mathsf{lm}$;
— $\mathsf{arOf}\ \mathsf{lam}\ i = \mathsf{undefined}$;
- $\mathsf{barOf} : \mathbf{opSym} \to \mathbf{Input}(\mathbf{bindex}, \mathbf{varSort} \times \mathbf{sort})$ is defined by:
— $\mathsf{barOf}\ \mathsf{app}\ i = \mathsf{undefined}$;
— $\mathsf{barOf}\ \mathsf{lam}\ i = \mathsf{lm}$.

Now, $\mathbf{term}(\Sigma, \mathsf{lm})$ and $\mathbf{abs}(\Sigma, (\mathsf{vlm}, \mathsf{lm}))$ are precisely the terms and abstractions of untyped λ-calculus discussed at the beginning of this section (with $\mathsf{Op}\ \mathsf{app}$ being precisely App, etc.)

For the LF syntax. We take the following binding signature Σ:
- since we have two syntactic categories of variables and three of terms,
— **varSort** is a two-element set, say $\{\mathsf{vo}, \mathsf{vt}\}$;
— **sort** is a singleton set, say $\{\mathsf{o}, \mathsf{t}, \mathsf{k}\}$;
— $\mathsf{asSort} : \mathbf{varSort} \to \mathbf{sort}$ maps vo to o and vt to t;

- $\mathbf{opSym} = \{\mathsf{app}, \mathsf{lam}, \mathsf{tapp}, \mathsf{tlam}, \mathsf{tprod}, \mathsf{type}, \mathsf{kprod}\}$;
- $\mathbf{index} = \{\mathsf{Iapp}_1, \mathsf{Iapp}_2, \mathsf{Ilam}, \mathsf{Itapp}_1, \mathsf{Itapp}_2, \mathsf{Itlam}, \mathsf{Itprod}, \mathsf{Ikprod}\}$;
- $\mathbf{bindex} = \{\mathsf{Ilam}, \mathsf{Itlam}, \mathsf{Ikprod}\}$

(Notice that, e.g., since lam is meant to take one free argument and one bound argument, it gets one slot in **index** and one slot in **bindex**.)

- stOf : $\mathbf{opSym} \to \mathbf{sort}$ is defined by:

— stOf app = o; stOf lam = o;

— stOf tapp = t; stOf tlam = t; stOf tprod = t;

— stOf type = k; stOf kprod = k;

- arOf : $\mathbf{opSym} \to \mathbf{Input}(\mathbf{index}, \mathbf{sort})$ is defined by:

— arOf app i = if $i \in \{\mathsf{Iapp}_1, \mathsf{Iapp}_2\}$ then o else undefined;

— arOf lam i = if i = Ilam then t else undefined;

— arOf tapp i = if $i = \mathsf{Itapp}_1$ then t elseif $i = \mathsf{Iapp}_2$ then o else undefined;

— arOf tlam i = if i = Itlam then t else undefined;

— arOf tprod i = if i = Itprod then t else undefined;

— arOf type i = undefined;

— arOf kprod i = if i = Ikprod then t else undefined;

- barOf : $\mathbf{opSym} \to \mathbf{Input}(\mathbf{bindex}, \mathbf{varSort} \times \mathbf{sort})$ is defined by:

— barOf app i = undefined;

— barOf lam i = if i = Ilam then (vo, o) else undefined;

— barOf tapp i = undefined;

— barOf tlam i = if i = Itlam then (vo, t) else undefined;

— barOf tprod i = if i = Itprod then (vo, t) else undefined;

— barOf type i = undefined;

— barOf kprod i = if i = Ikprod then (vo, k) else undefined;

Note that isInBar(xs, s) holds iff $(xs, s) \in \{(\mathsf{vo}, \mathsf{o}), (\mathsf{vo}, \mathsf{t}), (\mathsf{vo}, \mathsf{k})\}$, and therefore we have three categories of abstractions. Now,

- $\mathbf{term}(\Sigma, \mathsf{o})$, $\mathbf{term}(\Sigma, \mathsf{t})$ and $\mathbf{term}(\Sigma, \mathsf{k})$ are what we previously denoted by $\mathbf{term}_{\mathsf{o}}$, $\mathbf{term}_{\mathsf{t}}$ and $\mathbf{term}_{\mathsf{k}}$, respectively;
- $\mathbf{abs}(\Sigma, (\mathsf{vo}, \mathsf{o}))$, $\mathbf{abs}(\Sigma, (\mathsf{vo}, \mathsf{t}))$ and $\mathbf{abs}(\Sigma, (\mathsf{vo}, \mathsf{k}))$

are what we previously denoted by $\mathbf{abs}_{\mathsf{o},\mathsf{o}}$, $\mathbf{abs}_{\mathsf{o},\mathsf{t}}$ and $\mathbf{abs}_{\mathsf{o},\mathsf{k}}$, respectively.

2.9.3 Formalization of the general theory

The general setting for syntax with bindings sketched in Section 2.9.1 has been formalized in Isabelle/HOL. The Isabelle scripts presented in browsable html format can be downloaded from [120]. These scripts are fairly well documented by text inserted at the beginning of each theory/section and often at the beginning of subsections too. Next we give an outline

of this formalization. The relevant part of the theory structure is shown in Figure 2.1.

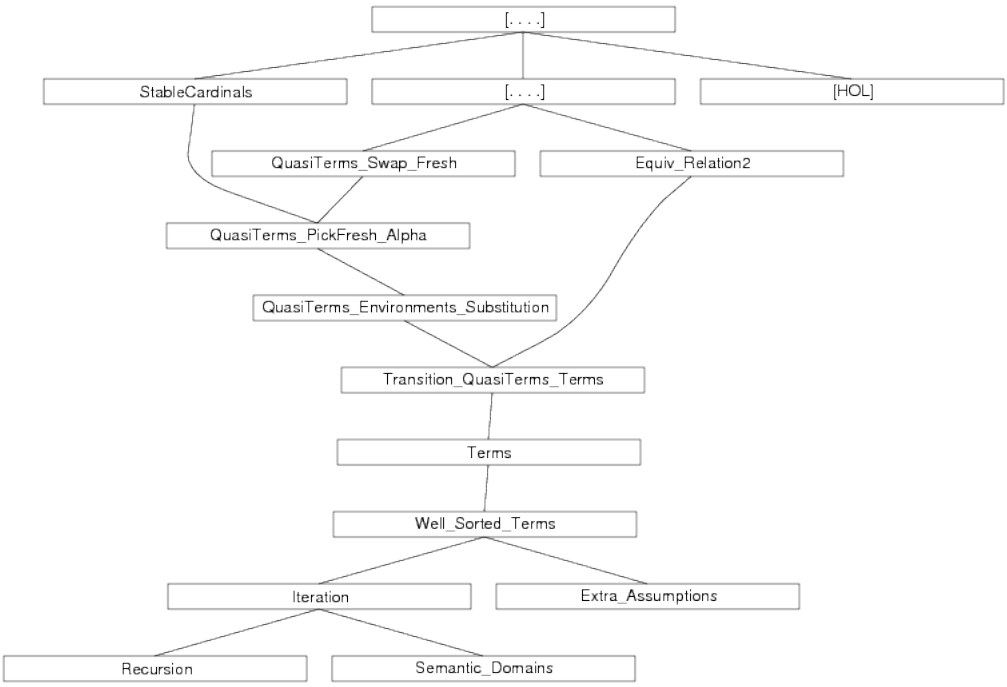

Figure 2.1: The Isabelle theory structure for general syntax with bindings

The theories StableCardinals and EquivRelation2 contain some preliminaries on cardinals and equivalence relations required for the development of the general theory of syntax (the former for dealing with possibly infininitary operators, the latter for factoring to α-equivalence).

Formalization of terms. The theories QuasiTerms_Swap_Fresh, QuasiTerms_PickFresh_Alpha and QuasiTerms_Environemnts_Substitution deal with quasi-items (i.e., quasi-terms, quasi-abstractions, etc.). They are introduced and studied in an *unsorted* format, by the following inductive definition:[16]

- Any quasi-term qX is:

— either a variable injection, qVar xs x;

— or an operation applied to a quasi-input and a quasi-binput, qOp δ $qinp$ $qbinp$.

- Any quasi-abstraction qA has the form qAbs xs x qX;

- A quasi-input $qinp$ is a partial function from indexes to quasi-terms;

[16] In the Isabelle scripts, for as long as we do not have "the true items" defined yet, we do not prefix the quasi-item meta-variables by "q", and thus write X, Y, Z for quasi-terms, etc.

- A quasi-binput *qbinp* is a partial function from bindexes to quasi-abstractions.
(In the scripts, the above comes as a *datatype* definition, at the beginning of the theory QuasiTerms_Swap_Fresh.)

Notice that so far the difference between the formalization and the definition of Σ-quasi-items from Section 2.9.1 is that the former are not classified according to sorts. In fact, the Isabelle types of quasi-terms and quasi-abstractions are not (and actually, due to the Isabelle type system, cannot be) parameterized by a signature Σ, but merely by the various types required to build "raw" (i.e., unsorted) quasi-items: **index**, **bindex**, **varSort**, **var** and **opSym**. The reason why we were not eager to "sort" in our formalization was the desire to keep the complexity as low as possible for as long as possible – indeed, part of the theory of syntax can be performed on unsorted items[17] *in such a way that it will be immediately transported to sorted items.*

α-equivalence on quasi-items is defined in the theory QuasiTerms_PickFresh_Alpha. In order for the necessary properties of α to hold (including the very fact that it is an equivalence), we had to make three assumptions, included as part of the Isabelle locale FixVars,[18] defined as the beginning of the mentioned theory:
- (1) the set **var** of variables is infinite;
- (2) the cardinality of **var** is stable;
- (3) there are fewer variable sorts than variables.

Above, assumption (1) is standard. Assumption (2) has to do with our desire to allow for infinitary operators too. Stability is a generalization of the notion of countability – the cardinal of the set $\mathbb{N}$ of natural numbers is stable because a finite union of finite sets if again finite. In general, a cardinal c is stable iff, for all families $(A_i)_{i \in I}$ where the cardinality of I and that of each A_i are less than c, it holds that the cardinality of $\bigcup_{i \in I} A_i$ is also less than c. (See theory StableCardinals from [120] for more details.) Assumption (3) is needed for ensuring the right behavior of parallel substitution. If a signature has a finite number of sorts and has all its operation symbols finitary (as do most signatures of interest in syntax with bindings), then taking **var** to be a copy of $\mathbb{N}$ satisfies the assumptions (1)-(3).

In the above context, *good quasi-terms* are defined to be essentially terms whose branching at each Op node are smaller than the cardinality of variables. This ensures that they do not exhaust all the variables, hence that it is possible to pick fresh variables for them. The theory of α-equivalence is then developed for good quasi-terms.

The theory Transition_QuasiTerms_Terms "transits" from quasi-terms and quasi-abstractions to terms and abstractions by defining the latter as α-equivalence classes of the former. More precisely, we have *good terms* and *good abstractions* as α-equivalence classes of good quasi-terms and good quasi-abstractions. Therefore, we have the following:

[17] More precisely, on what we call "good" items – see below.

[18] In Isabelle, a *locale* is a persistent context where one can fix several parameters and assumptions about them, and prove facts following from these assumptions.

- Any good term X is:
— either a variable injection, Var xs x;
— or an operation applied to a good input and a good binput, Op δ inp $binp$.
- Any good abstraction A has the form Abs xs x X;
- A good input inp is a partial function from indexes to terms whose domain is smaller than **var**;
- A good binput $binp$ is a partial function from bindexes to abstractions whose domain is smaller than **var**;
- Everything is "up to α", considering that, within Abs xs x X, (xs, x) is bound in X.

In the theory Terms we develop a rich mathematical theory for good terms and their the standard operations: freshness, swapping, (unary) substitution and parallel substitution. A useful particular case of substitution, variable-for-variable substitution, is also defined.

The theory Well_Sorted_Terms finally introduces sorting of terms according to a signature. Well-sorted terms are defined in a locale, FixSyn, which is an extension of the previous locale FixVars with the following:
- we fix a binding signature Σ (the notion of binding signature is faithfully the one introduced in Section 2.9.1);
- we make several assumptions about its components, namely:
— (1) the map asSort : **varSort** $\rightarrow$ **sort** is injective;[19]
— (2) the domain of each arity arOf δ is smaller than **var**;
— (3) the domain of each barity barOf δ is smaller than **var**;
— (4) the type **sort** of sorts is smaller than **var**.
(Again, these very general assumptions are satisfied by most syntaxes in current use.)

In the above context, well-sorted terms, abstractions, inputs and binputs are introduced as the predicates
- wls : **sort** $\rightarrow$ **term** $\rightarrow$ **bool**,
- wlsAbs : **varSort** $\times$ **sort** $\rightarrow$ **abs** $\rightarrow$ **bool**,
- wlsInp : **opSym** $\rightarrow$ **Input**(**index**, **term**) $\rightarrow$ **bool**,
- wlsBinp : **opSym** $\rightarrow$ **Input**(**bindex**, **abs**) $\rightarrow$ **bool**,
defined mutually inductively as follows:
- wls (asSort xs) (Var xs x);
- if wlsInp δ inp and wlsBinp δ $binp$, then wls (stOf δ) (Op δ inp $binp$);
- if isInBar(xs, s) and wls s X, then wlsAbs (xs, s) (Abs xs x X);
- if sameDom (arOf δ) inp and $\forall i\ X\ s.$ arOf δ i = Some s $\wedge$ inp i = Some $X \Longrightarrow$ wls s X,[20] then wlsInp δ inp;
- if sameDom (barOf δ) $binp$ and $\forall i\ A\ xs\ s.$ barOf δ i = Some (xs, s) $\wedge$ $binp$ i = Some $A \Longrightarrow$

[19] This assumption is also part of the definition of signature from Section 2.9.1.

[20] In Isabelle, partial functions to a type T are modeled as total functions to the type **Option**(T), which consists of copies Some t of the elements t of T and of the "undefined" value called None.

wlsAbs (xs, s) A, then wlsBinp δ $binp$.

Therefore, the set $\mathbf{term}(\Sigma, s)$ from Section 2.9.1 is the same as the set $\{X.\ \mathsf{wls}\ s\ X\}$ in our formalization, and similarly for:
- $\mathbf{abs}(\Sigma, (xs, s))$ versus $\{A.\ \mathsf{wlsAbs}\ (xs, s)\ A\}$,
- $\mathbf{inp}(\Sigma, \delta)$ versus $\{inp.\ \mathsf{wlsInp}\ \delta\ inp\}$,
- $\mathbf{binp}(\Sigma, \delta)$ versus $\{binp.\ \mathsf{wlsBinp}\ \delta\ binp\}$.

A technical note – lifting operators. To keep statements about inputs (and binputs) concise, we define the following polymorphic operators for lifting functions and predicates to inputs:
- lift : $(A \to B) \to \mathbf{Input}(I, A) \to \mathbf{Input}(I, B)$,
by lift f inp i = case inp i of Some $a \Rightarrow$ Some $(f\ a)$ | None $\Rightarrow$ None;
- liftAll : $(A \to \mathbf{bool}) \to \mathbf{Input}(I, A) \to \mathbf{bool}$,
by liftAll φ inp $i = (\forall i, a.\ inp\ i = \mathsf{Some}\ a \Longrightarrow \varphi\ a)$;
- $\mathsf{liftAll}_2$: $(A \to B \to \mathbf{bool}) \to \mathbf{Input}(I, A) \to \mathbf{Input}(I, B) \to \mathbf{bool}$,
by $\mathsf{liftAll}_2$ φ inp inp' $i = (\forall i, a, b.\ inp\ i = \mathsf{Some}\ a\ \wedge\ inp'\ i = \mathsf{Some}\ b \Longrightarrow \varphi\ a\ b)$.

Thus, lift is the expected "map" for inputs, liftAll checks if a predicate holds for all values of an input, and $\mathsf{liftAll}_2$ checks if a binary predicate holds for all synchronized values of two inputs. These operators are omnipresent throughout our scripts. For instance, the condition
- $\forall\, i\ X\ s.$ arOf $\delta\ i$ = Some $s\ \wedge\ inp\ i$ = Some $X \Longrightarrow$ wls $s\ X$
from a previous definition is written compactly as $\mathsf{liftAll}_2$ wls (arOf δ) inp.

Formalization of induction principles. In the absence of a depth operator (our terms being allowed to be infinitely branching), we base our induction principles on an operator giving the so-called "skeleton" of a term, which is a well-founded tree retaining only the branching information from the original term.[21]

We have structural induction principles with various flavors:
- Generalizations of Prop. 2.3, including the following theorems:
— wls_induct from theory Well_Sorted_Terms, employing, for the abstraction case, either of the skeleton, or the swapped relation, or variable-for-variable substitution (whatever the user finds convenient for a given situation);
— wls_induct_depth from theory Extra_Assumptions, a version of the above replacing the skeleton with the (at this point available) depth operator.
- Generalizations of Prop. 2.5 ("fresh" induction), including the following theorems from theory Well_Sorted_Terms:
— wls_templateInduct_fresh; this is the most general form, employing a general notion of parameter and also featuring a skeleton-preserving relation again for maneuvering in the

[21] See also the presentation of theory Extra_Assumptions later in this section.

abstraction case;
— wls_rawInduct_fresh, a version of the above without the extra relation;
— wls_induct_fresh, a version with the notion of parameter customized to consist of lists of variables, abstractions, terms and environments.[22]

Formalization of the recursion principles. The theories Iteration and Recursion formalize the arbitrary-syntax generalization of the results pertaining to Horn-based iteration and recursion with built-in substitution from Sections 2.5.1 and 2.7, as well as their variations from Section 2.8.

We use the prefix "g" for generalized items from the theory Recursion, and the longer prefix "ig" (where "i" stands for "iteration") for the generalized items from the theory Iteration. Moreover, what we call in the scripts fresh-substitution and fresh-swap models are actually their full-recursive variants; for the simpler, iterative variants, we use in the scripts the term "imodel". We prefer the shorter terminology and notation for full recursion since, being more general, is the only one we "keep" with us. (But iteration is used to infer full recursion in first place, as indicated in the proof sketch for 2.10.)

Next we only discuss the formalization of (full-recursion) models from the theory Recursion, but most of the discussion applies to the imodels from Iteration as well. We have four kinds of models:
- FSb-models, formalizing the concept described in Section 2.5.1;
- FSw-models, formalizing the swapping-based variation described in Section 2.8.1;
- FSbSw-models and FSwSb-models, formalizing the two substitution-swapping combinations described in Section 2.8.2.

For economy reasons, all these kinds of models share a common Isabelle record type, featuring operators for the syntactic constructs, freshness, swapping and substitution – such a record is called a "raw" model. The involved clauses, such as F1-F4, S1-S4 and AR for FSb-models, stated as predicates on raw models, make such a raw model a specific model of one of the 4 kinds; if a certain feature is not needed for a certain kind of model (such as swapping for FSb-models), the corresponding operator is left undefined.

More precisely, reflecting our general setting for terms (including the distinction between terms and abstractions), a (raw) model, defined at the beginning of theory Recursion, is a record, depending on the types **index**, **bindex**, **varSort**, **sort**, **opSym**, **var**, as well as on the types **gTerm** and gAbs of generalized terms and abstractions (the latter representing the carriers), and consisting of the following Isabelle constants:
- Well-sortedness predicates:
— gWls : **sort** → **gTerm** → **bool**;

[22]The last three theorems are listed in the decreasing order of their generality; remember that the goal in formal development is to state not only the most general facts, but also facts that are convenient to use; some less general facts are more convenient (if applicable).

— gWIsAbs : $\mathbf{varSort} \times \mathbf{sort} \to \mathbf{gAbs} \to \mathbf{bool}$.
- Generalized term and abstraction constructs:
— gVar : $\mathbf{varSort} \to \mathbf{var} \to \mathbf{gTerm}$;
— gAbs : $\mathbf{varSort} \to \mathbf{var} \to \mathbf{term} \to \mathbf{gTerm} \to \mathbf{gAbs}$;
— gOp : $\mathbf{opSym} \to \mathbf{Input}(\mathbf{index}, \mathbf{term}) \to \mathbf{Input}(\mathbf{index}, \mathbf{gTerm}) \to \mathbf{Input}(\mathbf{bindex}, \mathbf{abs}) \to \mathbf{Input}(\mathbf{bindex}, \mathbf{gAbs}) \to \mathbf{gTerm}$;
- Generalized freshness, swapping and substitution operators:
— gFresh : $\mathbf{varSort} \to \mathbf{var} \to \mathbf{term} \to \mathbf{gTerm} \to \mathbf{bool}$;
— gFreshAbs : $\mathbf{varSort} \to \mathbf{var} \to \mathbf{abs} \to \mathbf{gAbs} \to \mathbf{bool}$;
— gSwap : $\mathbf{varSort} \to \mathbf{var} \to \mathbf{var} \to \mathbf{term} \to \mathbf{gTerm} \to \mathbf{gTerm}$;
— gSwapAbs : $\mathbf{varSort} \to \mathbf{var} \to \mathbf{var} \to \mathbf{abs} \to \mathbf{gAbs} \to \mathbf{gAbs}$;
— gSubst : $\mathbf{varSort} \to \mathbf{term} \to \mathbf{gTerm} \to \mathbf{var} \to \mathbf{term} \to \mathbf{gTerm} \to \mathbf{gTerm}$;
— gSubstAbs : $\mathbf{varSort} \to \mathbf{term} \to \mathbf{gTerm} \to \mathbf{var} \to \mathbf{abs} \to \mathbf{gAbs} \to \mathbf{gAbs}$.

Then the four types of models are introduced by the predicates wIsFSb, wIsFSw, wIsFSbSw, wIsFSbSw, read "well-structured FSb-model" etc., which are conjunctions of the necessary clauses. Thus, e.g., given a raw model MOD, wIsFSb MOD is defined to be a conjunction of six predicates:
- gWIsAllDisj MOD, stating that the carriers of the model (given by gWIs and gWIsAbs) are disjoint for distinct sorts;
- gWIsAbsIsInBar MOD, a technical condition stating that only carriers of meaningful abstractions (namely, those for pairs (xs, s) for which the predicate isInBar(xs, s) holds) are non-empty;
- gConsPresGWIs MOD, stating that the generalized syntactic constructs preserve (i.e., are well-defined on) the carriers;
- gSubstPresGWIsAll MOD, stating carrier preservation by generalized substitution;
- gFreshCls MOD, stating "the fresh clauses" (generalizing F1-F4 from Section 2.5.1);
- gSubstCls MOD, stating "the substitution clauses" (generalizing S1-S4 from Section 2.5.1);
- gAbsRen MOD, stating "the abstraction-renaming clause" (generalizing AR from Section 2.5.1).

The expected notions of morphism from the term model to a model of each of the four kinds (generalizing the notion of FR-morphism from Section 2.5.1 and the corresponding ones for the variations) is given by the predicates termFSbMorph, termFSwMorph and termFSwSbMorph (there are only three of them, since the notions of morphisms for FSbSw and FSwSb models coincide), stating preservation of well-sortedness, freshness and of substitution and/or swapping, depending on the kind.

Given a FSb-model MOD as above, the associated recursive morphism from the term model comes as two maps rec MOD : $\mathbf{term} \to \mathbf{gTerm}$ and recAbs MOD : $\mathbf{abs} \to$ gAbs. These are defined as prescribed in the proof sketch for Th. 2.10, based on the iterator (dealt with

in the previous theory Iteration). The recursion theorems are then stated:
- The generalization of Th. 2.10, split in two:
— existence, as theorem wIsFSb_recAll_termFSbMorph (note that H from Th. 2.10 is given here by rec MOD and recAbs MOD);
— uniqueness, as theorem wIsFSb_recAll_unique_presCons (saying that (rec MOD, recAbs MOD) is the only pair of construct-preserving maps from terms and abstractions to the target model).
And similarly for the other three kinds of models.

Generalizations of the criteria for extra morphism properties from Section 2.7 have also been formalized:
- Th. 2.11, as theorem wIsFSb_recAll_reflFreshAll,
- Th. 2.12, as theorem wIsFSb_recAll_isInjAll,
- Th. 2.13, as theorem wIsFSb_recAll_isSurjAll.
And again, similarly for the other three kinds of models.

A difference between the presentation in Section 2.7 and the formalizated theorems pointed in the previous paragraph is that for the latter we use full-recursion models instead of iterative models – this is because later, in case-studies, we only wish to employ the notion of full-recursion models (iterative models being a particular case, while not more convenient to use). Consequently, in the surjectiveness criterion, theorem wIsFSb_recAll_isSurjAll, we need an extra hypothesis saying that the given full-recursion model has its syntactic constructs "indifferent" to the concrete-term (and concrete-abstraction) arguments, in other words, that the model under-behaves (w.r.t. the syntactic constructs) like an iterative model.

Some notes on engineering many-sortedness for iteration. As mentioned, we mainly care about the content of theory Iteration (iteration principles) in order to obtain full recursion, and then (beyond inferring semantic-domain interpretation – see below) we forget about it. However, the "hard work" behind the end-product recursion principles lays in this theory. In order to build the desired map to the target model MOD, we have essentially followed the path indicated in the proof sketch of Th. 2.9:
- (1) Start with the standard initial map (commuting with the syntactic constructs) from the free structure of quasi-terms to (the carrier of) MOD – this map, called qInit MOD in the scripts (and having qInitAbs MOD as its abstraction counterpart),[23] is given for free by the Isabelle datatype recursion mechanism.
- (2) Then prove the necessary preservation facts (of freshness, α-equivalence, etc.).
- (3) Finally, construct a map from terms (i.e., α-classes) to MOD by the universal property of quotients. This map, called iter MOD in the scripts, is the desired morphism, i.e., the desired iterator.

[23] In what follows, we shall omit mentioning the abstraction counterparts of the various maps.

There was a difficulty with step (3) in the above path. Recall that we have α as a congruence not only on well-sorted terms, but also on the (unsorted) good terms. Since using a many-sorted version of the universal property of quotients would be tedious to formalize (and such a formalization would have to take into account congruence issues too), we of course preferred to take advantage of the above flexibility of α and use the *unsorted* version of the aforementioned universal property. But the Horn axiomatization of models is sorted – e.g., a clause such as freshAbs xs x (gAbs xs gX) is not supposed to hold for any raw gX, but only for those of sort asSort xs. Indeed, being sort-flexible for concrete terms is harmless (and in fact helpful), but for axiomatizing models such flexibility works against us, as it triggers unnecessarily general facts that the user needs to prove to have an iterative definition go through. So for models we had to be "sort-strict", implying that step (3) would not go through w.r.t. unsorted good terms.

The solution was to introduce an intermediate, sort-flexible axiomatization, consisting of what we called in the scripts "the strong clauses", which are versions of the original clauses making no sort hypotheses. Then, starting with a model MOD for the original axiomatization, we constructed a model for the strong clauses, called errMOD MOD, by introducing an error element ERR on which we dumped all the results of applying operators to sort-violating arguments (for the case of the freshness relation, we dumped all such results onto True). The function check : errMOD MOD $\to$ MOD, mapping all the non-error elements (called "OK-elements" in the scripts) to themselves and mapping ERR no matter where (i.e., to an arbitrary element called undefined in Isabelle) is a morphism of models. Moreover, for the "strong model" MOD$'$ = errMOD MOD, step (3) works w.r.t. good terms, and in particular we obtain a morphism, called iterSTR MOD$'$, from well-sorted terms to this model. Finally, the composition of check and iterSTR MOD$'$ is our desired morphism, iter MOD.

Formalization of semantic-domain interpretations. Due to its importance and generality, the semantic interpretation example (or, rather, class of examples) discussed in Section 2.3.1 as motivational Problem I has also been included in the general development (for an arbitrary syntax) – this is the topic of theory Semantic_Domains. A "well-structured" semantic domain (predicate wlsSEM) consists essentially of a type **sTerm**, of semantic values (called "semantic terms" in the scripts' comments), and of an interpretation of the operation symbols from a signature, where abstractions are treated as functions (thus the operations have second-order arguments to match bindings in the syntax, as for LM versus Lm at Problem I). A compositional interpretation map (predicate compInt in the scripts) is a map, via valuations, of syntax to the semantic domain which is compositional with the syntactic constructs and with substitution and oblivious to freshness, as is [_] in Problem I. The main theorem, semIntAll_compInt, states that, for any given semantic domain SEM, the pair (semInt SEM, semIntAbs SEM), defined using the fresh-substitution-swapping (FSbSw) iterator

from theory Iteration, is indeed such a compositional interpretation map.[24]

Extra assumptions. Our binding signatures are quite general, e.g., as we have already mentioned, they allow infinitary operations. (Recall that, in our general theory, the role of justifying induction and recursion, standardly belonging to depth, is played by the aforementioned "skeleton" operator.) This generality is too much for most cases, and it also denies us access to some useful operators and facts. Therefore, in the theory Extra_Assumptions we have defined various locales extending our base locale FixSyn, where we make more commitments and prove some consequences of these commitments. Namely, we consider combinations of the following extra assumptions:
- (1) finite arities for the operation symbols;
- (2) finite number of sorts;
- (3) fewer operation symbols than variables;
- (4) varsorts the same as sorts.

The most important one is (1), which allows us to define the depth. Moreover, the combination of (1) and (3) allows us to infer some useful cardinality facts, such as not having more terms than variables (theorem wls_ordLeq_var).

2.9.4 Formalization of the examples and larger developments

All of the examples listed explicitly in this chapter have been formalized. As mentioned, our first example, namely, semantic interpretation (appearing as Problem I in Section 2.3.1), has been included as a "built-in" in the general theory.[25] The rest of the examples are formalized in roughly the same syntax in which they were presented in this chapter (usually in a slightly more general format, allowing an unspecified number of constants too besides variables, application and λ-abstraction).

Many of the examples appear in the formal scripts as parts of larger developments, the largest one being a formalization of a significant part of Plotkin's classic paper [117]. In all such cases, we claim that the overall development benefits highly from the possibility to *define the map with the desired properties and move on*, provided by our recursion principles, as opposed to trying to make the definition work by ad hoc bases.

Next we give an overview of the various λ-calculus constructions and results we have formalized, with an emphasis on the places where we used our recursion principles. The formal scripts, including all the theories we discuss below, as well as all the underlying general theory described in Section 2.9.3 and available separately at [120] (in other words,

[24] We preferred to employ the richer FSbSw-iteration, instead of just FSb-iteration as discussed in Section 2.5.2, since semantic domains also have a natural and potentially useful notion of swapping with which the interpretation commutes.

[25] But its concrete instance for the λ-calculus can be seen in the theory L described below.

all the formal scripts pertaining to the first part of this dissertation) are available at [121]. Again, all these scripts are well-documented by text inserted at the beginning of each theory and often at the beginning of each theory subsection too.

Instances of the general theory for particular syntaxes. So far, we have considered the following two instances:
- (1) The syntax of the untyped λ-calculus with constants, with terms $X, Y, Z \in \mathbf{term}$ and abstractions $A, B \in \mathbf{abs}$:

$$\begin{array}{lcl} X & ::= & \mathsf{Var}\ x \mid \mathsf{Ct}\ c \mid \mathsf{App}\ X\ Y \mid \mathsf{Lam}\ A \\ A & ::= & \mathsf{Abs}\ x\ X \end{array}$$

We let $\mathsf{Lm}\ x\ X$ denote $\mathsf{Lam}\ (\mathsf{Abs}\ x\ X)$.

- (2) The two-sorted value-based variation of the above, with full terms $X, Y, Z \in \mathbf{term}_{\mathit{fl}}$, value terms $Xvl, Yvl, Zvl \in \mathbf{term}_{\mathit{vl}}$ and abstractions (of value variables in full terms) $A, B \in \mathbf{abs}_{(\mathit{vl},\mathit{fl})}$:

$$\begin{array}{lcl} X & ::= & \mathsf{InVl}\ Xvl \mid \mathsf{App}\ X\ Y \\ Xvl & ::= & \mathsf{Var}\ x \mid \mathsf{Ct}\ c \mid \mathsf{Lam}\ A \\ A & ::= & \mathsf{Abs}\ x\ X \end{array}$$

where InVl is the injection of value terms into full terms. (Recall that this value-based variation is discussed in Section 2.6.3.) Here, too, we let $\mathsf{Lm}\ x\ X$ denote $\mathsf{Lam}\ (\mathsf{Abs}\ x\ X)$. Notice that here we only have value variables, hence it only makes sense to substitute value terms for variables.

The theories L1 and L are performing the instantiation of the general theory to the above syntax (1), and LV1 and LV do the same for (2). The instantiation process is performed in a completely uniform manner and will be eventually be automated, but currently it is done by hand.

Next we describe the contents of L1 and L (that of LV1 and LV being similar). L1 is essentially taking the actions described in Section 2.9.2:
- (a) defining the particular signature needed here, i.e., instantiating the locale FixSyn;
- (b) checking that the FixSyn assumptions are satisfied.

These steps are rather immediate, and they indeed give us the theory of the untyped λ-calculus. However, for the sake of convenience of reasoning for the given concrete syntax, we also take a third step, which requires more work:
- (c) transporting all the structure and theorems from the general-purpose format involving operation symbols and inputs as families to a more Isabelle-"native" setting involving:
— separate Isabelle types for each syntactic category (here, just two: terms and abstractions);

— concrete n-ary operators on terms, e.g., $\mathsf{App} : \mathbf{term} \to \mathbf{term} \to \mathbf{term}$, so that we can write $\mathsf{App}\ X\ Y$ instead of $\mathsf{Op}\ \mathsf{app}\ inp$, where inp is the two-element partial family containing X and Y.

While, to some extent, the above transport could have been performed by some abbreviations instead of actually defining new types and operations, we preferred the latter approach because of the higher degree of integration into the Isabelle typing system. (To help this tedious process of transport, we have prepared the general theory, by considering "in advance" some of the most encountered arities for concrete syntactic operators and replacing, for such cases, the input notation with the more convenient n-ary operation notation (without sacrificing generality though) – this is done in the "interface" theories Inter and Inter.)

Theory L contains all the facts one needs to know about the λ-calculus instance of the theory, including:
- (1) basic facts about freshness, substitution and swapping;
- (2) induction principles;
- (3) the Horn-based recursion principles.

All these facts, especially those pertaining to (3) are very well documented by comments inserted in between the formal scripts. Note that, in order to comprehend and use this theory, no knowledge of the formalization of the general theory is needed, since everything has been transported (in L1) to this specific syntax.

The theory of call-by-name λ-calculus. This is the content of the theories CBN and CBN_CR. We mainly follow Section 5 of [117].

In theory CBN, we introduce all the necessary relations pertaining to $\beta\delta$-reduction[26] – one-step left reduction, one-step reduction, parallel reduction, their reflexive-transitive closures, the associated CBN equational theory, etc. – as inductive definitions, and then prove several basic facts about them, such as fresh induction and inversion rules. (Note that some of the facts we prove by hand about these relations, namely those of the kind discussed in Section 2.2.2, would be available quasi-automatically if defined with the Nominal Package.) In proving these properties, we did not have in mind economy (i.e., proving only the necessary facts for our goal), but rather exhaustiveness (so to offer a "maximal" set of handles to these relations for any future development based on them).

The number-of-free-occurrences operator $\mathsf{no} : \mathbf{term} \to \mathbf{var} \to I\!N$ discussed in Section 2.6.1 plays an important role in the development, where a measure based on it needs to be assigned to parallel reduction (this measure roughly indicates the number of redexes available in parallel in one step).

In theory CBN_CR, we prove the call-by-name Church-Rosser theorem (which is a prerequisite for the results from [117]). We use the elegant method of Takahashi [145], based

[26] δ refers to the presence of the so-called δ rules for reducing constants application.

on the complete development operator cdev : **term** → **term** described in Section 2.6.6 (and resolved at the end of Section 2.8.1). Notice that our formalization is rather faithful to [145], in that cdev is indeed defined recursively *as a function* on terms. By contrast, other formal proofs of Church-Rosser based on complete development, including the Twelf [6] and Abella [4] proofs (available from their websites indicated in our citations) and the Isabelle Nominal proof (available in the Nominal directory of the Isabelle distribution) define cdev as a binary relation between terms, which complicates the simple proof presented in [145].

The theory of call-by-value λ-calculus. This is the content of the theories CBV and CBV_CR.We mainly follow Section 4 of [117], and the development has a similar structure to the one for call-by-name λ-calculus.

HOAS representation of λ-calculus into itself. This is the content of the theory HOAS. The first part of this theory deals with the representation of syntax; this is where we apply our recursion principle, as discussed in Sections 2.3.2 and 2.5.2.

To give the readers not interested in inspecting our Isabelle scripts an idea on how the formalization of such recursive definitions proceeds, we next list almost exhaustively the screenshots of this formal definition, and comment them. We work in a setting slightly more general than that mentioned in Section 2.3.2. Namely, recalling that **term**(**const**) denotes the syntax of the lambda-calculus over a set of constants **const**, we have the following:
- The object syntax is **term**(**const**).
- The meta syntax is **term**(**metaConst**), where the set **metaConst**, of *meta-constants*, consists of copies Const c of elements $c \in$ **const** and of the new constants APP and LM.

We use the prefixes "Obj" and "Meta" for operators pertaining to the object syntax and to the meta syntax, respectively. Moreover, we write $\_[\_/_o\ \_]$ and $\_[\_/_m\ \_]$ for the object level and meta level substitutions, respectively. The definition of the desired model, called rep_MOD, is the following:

```
definition rep_MOD ::
('const, 'const metaConst term, 'const metaConst abs)model
where rep_MOD ==
  gWls_lm = % mX. True,
  gWls_lm_lm = % mA. True,
  gVar_vlm = Meta_Var,
  gAbs_lm_lm = % y X mX. Meta_Abs y mX,
  gCt = % c. Meta.Ct (Const c),
  gApp = % X mX Y mY. Meta.App (Meta.App (Meta.Ct APP) mX) mY,
  gLam = % A mA. Meta.App (Meta.Ct LM) (Meta.Lam mA),
  gFresh_vlm_lm = % y X mX. Meta_fresh y mX,
```

```
gFresh_vlm_lm_lm = % y A mA. Meta_freshAbs y mA,
gSwap_vlm_lm = undefined,
gSwap_vlm_lm_lm = undefined,
gSubst_vlm_lm = % Y mY y X mX. mX [mY /m y],
gSubst_vlm_lm_lm = % Y mY y A mA. mA [mY /m y]
```

Explanations for the above:

- The type of rep_MOD (indicated on the second line) is the record type of "models for the **term**(**const**) syntax", hence the first (actual) parameter **const**; the other two parameters from this type represent the carriers of the model, which are the meta-terms and meta-abstractions.
- The lines starting from the fourth one define the components of the record, i.e., the "generalized operators" on the model.
- % is a notation for the Isabelle functional abstraction.[27]
- X, Y, A refer to object terms and abstractions, and mX, mY, mA to meta terms and abstractions.
- The various "lm" and "vlm" suffixes from the generalized-operators' names are a reminiscence of the many-sorted setting: they refer to the only varsort and (term) sort for the particular syntax **term**(**const**).
- The two "gWls" operators are the well-structuredness predicates for terms and abstractions, allowing one to select only part of either of these types as the carriers for the model; the flexibility offered by these operators is not needed here, so we take them to be vacuously true.
- Remember that we use one single record type, of "raw" models, for holding different kinds of models, featuring swapping and/or substitution operators. Here, we wish to employ FSb-models, not caring about swapping, hence we leave the latter undefined.

Modulo these explanations, and modulo the distinction we make between terms and abstractions in our formalization, the reader should now recognize the definition from the second part of Section 2.5.2. For instance, the above definition of gAbs_lm_lm corresponds to the following HOAS-specific recursive clause:

- $\mathsf{repAbs}\ (\mathsf{Obj.Abs}\ x\ X) = \mathsf{Meta.Abs}\ x\ (\mathsf{rep}\ X)$,

where rep denotes, as in Sections 2.3.2 and 2.5.2, the representation map on terms, and repAbs is its abstraction counterpart. Note that, since we employ full-recursion models, the object-syntax argument X is also available in definition of gAbs_lm_lm, but not used, since iteration suffices for this example.

In each of the Isabelle facts listed below, after the keyword "lemma",

- first comes the name of the lemma,

[27] λ is an alternative notation, but we preferred using ASCII symbols only to obtain the screenshot.

- then the actual statement in double quotes,
- and finally the proof.

Once the model is defined, we need to check that it is indeed a "well-structured" FSb-model, i.e., that it satisfies the involved clauses for freshness and substitution. In this case, Isabelle is able to check these automatically[28] (after we unfold the definitions):

```
lemma wlsFSb_rep_MOD:
"wlsFSb rep_MOD"
unfolding wlsFSb_defs by auto
```

This is all we needed to check to have our recursive definition go through. However, we wish to infer some further facts for our morphism, in this case freshness reflection and injectiveness. For these, Ths. 2.11 and 2.12, appearing in theory L1 as "wlsFSb rec refl_freshAll" and "wlsFSb rec is_injAll", respectively (which in turn are λ-calculus instances of the general theorems from theory Recursion, wlsFSb_recAll_reflFreshAll and wlsFSb_recAll_isInjAll, respectively) tell us that it suffices that the reversed fresh clauses hold and that the generalized constructs be injective. Again, Isabelle discharges these goals automatically:

```
lemma gFresh_cls_rev_rep_MOD:
"gFresh_cls_rev rep_MOD"
unfolding gFresh_cls_rev_defs by simp


lemma gCons_inj_rep_MOD:
"gCons_inj rep_MOD"
unfolding gCons_inj_defs by simp
```

This concludes the creative part of the development. The rest is only "bureaucracy", and goes in the same way for all our recursive definitions. As mentioned, we call rep and repAbs the associated recursive maps, between object terms and abstractions to the meta terms and abstractions:

```
definition rep where
"rep X == rec_lm rep_MOD X"


definition repAbs where
"repAbs A == rec_lm_lm rep_MOD A"
```

[28] Of course, "Isabelle" is not alone in this – our rich collection of simplification rules from theory L contribute the automation. Note, however, that these simplification rules are not tailored to solving a specific problem for a specific syntax, but are instances of facts holding for an arbitrary syntax.

It now remains to apply the actual recursion theorems, relating the desired properties of the recursive map with the facts we already proved about the model, obtaining that the pair (rep, repAbs) is a morphism which is additionally freshness reflecting and injective:

```
lemma term_FSb_morph_rep:
"term_FSb_morph rep repAbs rep_MOD"
unfolding rep_def_raw repAbs_def_raw
using wlsFSb_rep_MOD by(rule "wlsFSb rec term_FSb_morph")

lemma refl_freshAll_rep:
"refl_freshAll rep repAbs rep_MOD"
unfolding rep_def_raw repAbs_def_raw
using wlsFSb_rep_MOD gFresh_cls_rev_rep_MOD by(rule "wlsFSb rec refl_freshAll")

lemma is_injAll_rep:
"is_injAll rep repAbs"
unfolding rep_def_raw repAbs_def_raw
using wlsFSb_rep_MOD gCons_inj_rep_MOD by(rule "wlsFSb rec is_injAll")
```

Unfolding the involved definitions, we can see the desired facts in familiar format:

```
lemma rep_simps[simp]:
"rep (Obj_Var x) = Meta_Var x"
"rep (Obj.Ct c) = Meta.Ct (Const c)"
"rep (Obj.App X Y) = Meta.App (Meta.App (Meta.Ct APP) (rep X)) (rep Y)"
"rep (Obj_Lm y X) = Meta.App (Meta.Ct LM) (Meta_Lm y (rep X))"
using term_FSb_morph_rep unfolding term_FSb_morph_defs by simp_all

lemma rep_subst[simp]:
"rep (X [Y /o y]) = (rep X) [(rep Y) /m y]"
using term_FSb_morph_rep unfolding term_FSb_morph_defs by simp_all

lemma rep_preserves_fresh:
"Obj_fresh y X  ==>  Meta_fresh y (rep X)"
using term_FSb_morph_rep unfolding term_FSb_morph_defs by simp_all

lemma rep_reflects_fresh:
"Meta_fresh y (rep X) ==> Obj_fresh y X"
using refl_freshAll_rep unfolding refl_freshAll_defs by simp
```

```
corollary rep_fresh[simp]:
"Meta_fresh y (rep X) = Obj_fresh y X"
using rep_reflects_fresh rep_preserves_fresh by blast

lemma rep_inj[simp]:
"(rep X = rep Y) = (X = Y)"
using is_injAll_rep unfolding is_injAll_defs by auto
```

Thus,
- lemma rep_simps corresponds to clauses (1)-(3) from Section 2.3.2;
- lemma rep_subst is clause (4) from Section 2.3.2;
- corollary rep_fresh and rep_inj are some extra "built-ins" of the definition: preservation and reflection of freshness and injectiveness.

NB: Above, we have presented *the whole* formal development for defining the representation and proving its syntactic adequacy, except for a lemma, rep_MOD_simps (appearing in the scripts immediately after the definition of the model), which merely declares the definitions of the different components of the record as simplification rules. In particular, we did not omit listing any formal proofs – this is to give an idea of the degree of automation inherent in our recursion theorems.

The second part of the theory HOAS deals with the adequate representation of operational semantics (and has nothing to do with our recursion principles). We consider **term**(**metaConst**) as a logical framework which normalizes modulo $\beta\delta$ (i.e., according to the $\beta\delta$-reduction from [117]) on the background. This is close to the behavior of LF [63], except that:
- current implementations of LF consider η too in the normalization;
- LF reduction is guaranteed to reach a normal form, not the case for **term**(**metaConst**); however, the images through rep of object-level terms are normal forms; in fact, they are *strong normal forms*, meaning they are normal forms and stay so after substituting their variables with any normal forms.

As operational semantics for the object system we have chosen the call-by-name big-step reduction (as defined, e.g., in [117] on page 145, using the function Eval_N) – this is defined in our theory CBN as the relation Bredn : **Obj.term** $\to$ **Obj.term** $\to$ **bool**, with concrete syntax $==\!>_n$, where the index n is not a number, by a reminder of the "by name" style of reduction. (The choice was not important, we could have chosen any other reduction relation.) This relation is represented in the usual HOAS, LF-style fashion. The difference from LF though is that the simple logical framework considered here, **term**(**metaConst**), does not have judgement mechanisms of its own, and therefore we use the inductive mechanism of Isabelle, which is here the "meta-meta level". Thus, we represent Bredn by an inductively

defined relation MetaBredn : **Meta.term** → **Meta.term** → **bool**, with concrete syntax $\Longrightarrow_{nM}$, where the index "nM" stands for "by name, Meta". E.g., the β-clause in the definition of Bredn, namely,

$$\frac{X \Longrightarrow_n \mathsf{Obj.Lm}\ y\ Z' \qquad Z'[Y/y] \Longrightarrow_n U''}{\mathsf{Obj.App}\ X\ Y \Longrightarrow_n U''}$$

is captured by the clause

$$\frac{mX \Longrightarrow_{nM} \mathsf{Meta.App\ LM}\ mV' \qquad \mathsf{Meta.nf}\ (\mathsf{Meta.App}\ mV'\ mY) \Longrightarrow_{nM} mU''}{\mathsf{Meta.App}\ (\mathsf{Meta.App\ APP}\ mX)\ mY \Longrightarrow_{nM} mU''}$$

where, as usual, we omitted spelling the meta-constant injection operator, Meta.Ct, and where Meta.nf : **Meta.term** → **Meta.term** associates to each meta-level term its $\beta\delta$-normal form if such a normal form exists and is undefined otherwise.

Notice that the meta-level rule does not involve any bindings – this was the whole purpose, to capture the involved bindings implicitly, by the meta-level mechanisms. The object-level action of substituting in Z' a previously bound variable (as $Z'[Y/y]$) is matched by meta-level application Meta.App mV' mY in conjunction with normalization – since in this correspondence Meta.App LM mV' will be the representation of Obj.Lm y Z', mV' will itself be a (meta-level) Lm-abstraction; the purpose of normalizing Meta.App mV' mY is therefore taking care of the resulting meta-level β-redex.

For this sample representation, we have chosen to normalize "on the fly", since this matches most faithfully the practice of logical frameworks. (For instance, in (generic) Isabelle [155], $(\lambda\ x.\ X)\ Y$ is a volatile entity, being instantaneously replaced by $X[Y/y]$, the latter being what the user sees.) Another approach would be not normalizing (i.e., not including Meta.nf in the above rule), but then stating the adequacy referring to normal forms – the latter is more common in the LF literature on adequacy [63, 6]. Yet another, newer approach, surveyed in [64], is to normalize at substitution time, via a notion of *hereditary substitution*. These approaches are equivalent, and they reflect the semantic intuition that the β (or $\beta\delta$, $\beta\eta$, etc.) equational theory is acting as the meta-level equality.

The semantic adequacy of the representation is stated as two theorems:

- rep_preserves_Bredn: If $X \Longrightarrow_n X'$, then rep $X \Longrightarrow_{nM}$ rep X'.
- rep_reflects_Bredn: If rep $X \Longrightarrow_n mX'$, then there exists X' such that $X \Longrightarrow_n X'$ and rep $X' = mX'$.

In particular, these theorems imply:

- corollary rep_Bredn_iff: rep $X \Longrightarrow_{nM}$ rep X' holds iff $X \Longrightarrow_n X'$ holds.

Pure terms. Pure terms are terms not containing any constants. They form are a useful subset, since many λ-calculus developments ignore constants. The theory Pure defines them,

employing the function ctFresh from Section 2.6.4.

Connection with the de Bruijn representation. This is the content of the theory C_deBruijn. The scripts follow quite closely the notations we used in Section 2.6.2. We import theory Lambda, the existing formalization of the de Bruijn representation from the Isabelle library. Then we develop some de Bruijn arithmetics facts, after which we proceed with the interpretation map toDB by means of the model toDB_MOD. (This naming convention, having the name of the model be the name of the desired map followed by "_MOD" is observed in all our formalized examples.) This time, since there are some nontrivial computations involved in the verification of the clauses, we prefer to first define the model components separately (operators ggWls, ggVar etc. in the scripts). Lemmas toDB_simps, toDB_preserves_fresh and toDB_subst are the end-products of the definition, i.e. correspond to clauses (1)-(5) from Section 2.6.2. (For these end-products, we also observe the same naming pattern, as seen here and in the HOAS case, for all our examples.)

CPS transformation of call-by-value to call-by-name. This is the content of the theory CPS. It includes formalization of the discussion from Section 2.6.3. CPS imports the theory Embed, where the two-sorted value-based syntax is embedded in the (single-sorted) standard syntax.

Here, the main technical overhead was somewhat orthogonal to the usual problems with recursive definitions involving bindings, and it was about the choice of the fresh variables for the continuation combinators, choice itself not involved in the recursive mechanism. Thus, for the clauses listed at the beginning of Section 2.6.3, in [117] one simply decides to isolate the necessary fresh names, k, x, y, requiring that they never be used in the source terms. Then, e.g., in the clause

- cps (Lm x X) = Lm k (App k (Lm x (cps X))),

variable-capture is not a problem, since k is from a different universe than x and X. Suddenly deciding to single out some variables is reasonable in an informal mathematical discussion, but not in a formalization, especially since this non-uniform step is not really necessary. Indeed, a clause such as the above makes perfect sense for a regular fresh variable k, and, moreover, does not depend on the choice of k. In the scripts, we have defined the two involved combinators,

- combIC : **term** $\to$ **term**, read "the identity-continuation combinator",
- combAC : **term** $\to$ **term** $\to$ **term**, read "the application-continuation combinator",

corresponding to the right-hand sides of the clauses (3) and (4) from the end of Section 2.6.3, by making choices of fresh variables (via the Isabelle Hilbert choice) and then showing that these choices are irrelevant. Thus, the aforementioned clauses appear in the scripts (in lemma cps_simps) as:

- (3) cps(InVl Xvl) = combIC (cps$_{vl}$ Xvl)),
- (4) cps (App X Y) = combAC (cps X) (cps Y).

The "built-in" freshness reflection and injectiveness of the morphism are also formalized, as lemmas with names following our naming convention: cps_reflects_fresh and cps_inj.

The theory More_on_CPS then formalizes the results from the first part (call-by-value to call-by name) of Section 6 in [117] culminating with the theorems 1-3 listed on page 146 in op. cit.: Indifference, Simulation and Translation.

2.9.5 Certifying and relating various implementations of terms

The literature on λ-calculus theory and theorem proving abounds in approaches for representing syntax with bindings, such as α-classes [20], de Bruijn levels [36], de Bruijn indexes [36], locally nameless [57, 58, 13], locally named [139, 118], proper weak-HOAS functions [38, 61], partial functions [148] (to list only a few).

These representations have various merits: of being more or or less efficient, or more or less easy to implement, or more or less insightful w.r.t. definition and proof principles, etc. However, all these approaches are aimed at capturing the same notion of syntax. So what makes them correct? Typically, work proposing such a new implementation/representation justifies its correctness by showing it isomorphic to a more standard representation, or to one that has been used many times and has a high degree of trustability. This justification may be left informal, as in [148], where the new partial-function based representation of λ-terms from the Nominal package is proved isomorphic to the α-class-based one. But it may also be carried out formally, as is the case with the locally named representation from [118], justified by a formal isomorphism to the nominal representation.

Why should one trust a new representation? It is actually the case that we, personally, do trust the aforementioned representations, but this is *not* mainly because they were related to other, "clearly trustworthy" ones, but rather because, perhaps while building this relationship or as lemmas for different tasks, the authors *have proved enough familiar facts about their own representation*. This seemingly vague intuition can actually be made rigorous, and indeed formal, by characterization theorems such as our Th. 2.14, "enough familiar facts" meaning: a collection of facts that should clearly hold for the correct terms and should identify uniquely any model with the fundamental operators on it. In our opinion, there are two lists of candidates for these fundamental operators (at least as far as first-order operators go):[29]

- one includes the syntactic constructs, freshness and substitution;
- the other also includes swapping.

An argument for excluding swapping from the fundamental list is that the specification

[29] An extended list would also include HOAS operators.

of the relevant systems (typing systems, reduction systems etc.), as well as the (statements of) their main theorems, do not employ this operator. An argument for including swapping is that it is a very useful auxiliary operator in proofs. We have formalized (and experimented with) both options, by the characterizations of the FSb-, FSw-, FSbSw- and FSwSb- models[30] from Th. 2.7 (for FSb-models) and the like, formalized in the general theory as discussed in Section 2.9.3. To illustrate the usefulness of this formalization effort (which took a sensible amount of extra work beyond the strict purpose of having recursion principles), we have used it to certify formalizations of the λ-calculus different than ours.

While the fact that these formalizations would "pass our tests" was quite expected, there was a useful consequence of submitting them to these tests (along the lines of the traditional approach to certifying terms reviewed above): we obtained isomorphisms between our representation and theirs, thus creating a formal bridge on which results can be "borrowed" across different formalizations (much like it is prescribed in [28]). By "isomorphism", we mean a term-construct and substitution preserving, freshness [preserving and reflecting] (and perhaps also swapping preserving) bijection between our terms and theirs, based of course on a bijection between our variables and theirs. We have established such bridges between our representation and each of the following:

- (1) the Nominal representation [148];
- (2) the locally nameless representation underlying the Hybrid system [13, 92, 95, 43];
- (3) the so-called *locally named* representation from recent work [139, 118].

(1) The connection with Nominal is formalized in the theory C_Nominal. We connect the λ-terms from the theory Lam_funs (located in the Nominal directory in the Isabelle distribution) with our *pure* λ-terms. The latter are λ-terms without constants, defined in our theory Pure. (We need to use pure terms, since the terms in the aforementioned Nominal theory do not include constants.) First, we establish a bijection toNA, read "to Nominal atom" between our variables and the Nominal atoms, called "names" here – this is immediate, since both collections are countable. Then we proceed to organize the Nominal terms as an FSbSw-model with the required additional properties, yielding an isomorphism toN, read "to Nominal", between our pure terms and their terms. Checking the facts necessary for obtaining this isomorphism was immediate, given that Lam_funs and the underlying Nominal package provide a rich pool of basic facts.

To give an example of the potential usefulness of the above isomorphism, for both parties:

- our rich theory of substitution (including the recursion principles themselves) are now available for this Nominal theory, which does not excel in handling substitution, an operator external to the package;
- advanced Nominal techniques pertaining to rule induction are now available in our λ-calculus

[30] These four (highly overlapping) notions of models are still in an experimental stage w.r.t. their usefulness. It is probable that, after considering more case studies, we'll stick to only one of the above.

theory, which lacks at this aspect.

(2) For the connection with the Hybrid terms, we imported the theory Expr from the Hybrid scripts available online at http://hybrid.dsi.unimi.it. On top of Expr, Hybrid has several other HOAS layers – however, we were only interested in this very basic implementation layer. The connection, developed in our theory C_Hybrid, took a little more work, given the fact that the Hybrid representation is not FOAS-oriented, but HOAS-oriented. Consequently, we had to define ourselves a FOAS binding operation on Hybrid terms, LM. Then the Hybrid terms were organized as an FSb-model, yielding the desired isomorphism between our terms with constants and their terms with constants.

(3) The locally named representation from [118] is perhaps the most efficient one from the literature (in that it needs the smallest overhead in order to obtain the set of terms and its relevant operators). It is based on a distinction between parameters (or global variables), and (local) variables, only the latter being allowed (and required) to be bound. (More details can be found in [118], as well as in our commented scripts.) As mentioned, this representation has already been proved isomorphic to the Nominal one discussed above, by defining a relation and showing it to be total and deterministic, yielding a function – recall that a main motivation of our approach is to avoid the roundabout route of defining a function first as a relation. The connection is developed in the theory C_Sato_Pollack (named after the two authors of the approach), importing a couple of theories from the scripts associated to the paper [118], available from the first author's home page: http://homepages.inf.ed.ac.uk/rpollack. Similarly to before, we define a bijection toP between our variables and their parameters, and then organize their terms as an FSb-model, yielding an isomorphism, toCLN, read "to canonically named term" (this is the authors' terminology for their representation) between our pure terms and their terms. Again, proving the desired facts goes very smoothly, given that the authors have proved enough syntactic facts about terms.

2.10 Related work

Here we discuss some work related mainly to our main contribution in this chapter: Horn-based recursion for syntax with bindings.

2.10.1 On Horn recursion for finite sets

This topic is of course only tangent to that of our dissertation (and by no means are we trying to be exhaustive here). Yet, the work we refer to next is relevant because it explores a problem similar to ours and reaches similar conclusions. [102] discusses folding operators for finite sets in some detail (the context is Isabelle formalization, but the mathematical discussion is formalization-free). Their justification of the various presented principles is partly algebraic,

referring to *algebraic signatures*, which are Horn signatures without relation symbols. They distinguish two algebraic signatures, corresponding to two standard approaches to defining functions on finite sets (on page 387 in op. cit.): that of $(\emptyset, \{-\}, \cup)$-algebras, and that of $(\emptyset, \mathsf{insert})$-algebras – the former is the signature of semilattices that we reviewed briefly in Section 2.4.2, and the latter is the algebraic part of the FOL signature Σ_S we discussed in more detail when defining card in the same section.

Among the considered examples are sums (and, similarly, products) over finite sets, $\mathsf{setsum} : (\mathbf{item} \to I\!N) \to \mathbf{P}_f(\mathbf{item}) \to I\!N$, with $\mathsf{setsum}\ f\ \{i_1, \ldots, i_n\}$ being computed by folding $+$ over $\{i_1, \ldots, i_n\}$, i.e., as $f\ i_1 + \ldots + f\ i_n$ (if all $i_1, \ldots, i_n$ are distinct). In the conclusion of op. cit., the authors comment that the setsum and set product examples "are not homomorphisms, but still definable". Interestingly, the only pieces missing from the puzzle in order to regard such cases, and, indeed, all the examples from op. cit., as (homo)morphisms too (and thus subsume them to a common uniform principle) are the extra bits of generalization advocated in this chapter:
- from universal algebra to Horn theory on one hand;
- from standard models to full-recursion models on the other.

Indeed, if one switches from $(\emptyset, \mathsf{insert})$-algebras to our Σ_S-FR-models, then the clauses:
- $\mathsf{setsum}\ f\ \emptyset = 0$,
- $\mathsf{setsum}\ f\ (\{i\} \cup s) = (\mathsf{setsum}\ f\ s) + 1$, if $i \notin s$

can be seen as defining $\mathsf{setsum}\ f$ as the unique FR-morphism to a corresponding FR-model (the $\boxed{\mathsf{Emp}}$- and $\boxed{\mathsf{Insert}_i}$- operators are defined in the obvious way to match the above clauses, and the relation $\boxed{\mathsf{fresh}_i}$ is defined similarly to the case of card).

(It is also possible to extend the signature and algebraic theory of $(\emptyset, \{-\}, \cup)$-algebras to a Horn theory and base the definition of setsum on it, but this would be less convenient than the above.)

2.10.2 On recursion for syntax with bindings

Nominal Logic. This is a very influential FOAS approach to syntax with bindings [115]. More specifically to the topic of this chapter, [116] is a fairly recent detailed account of Nominal induction and recursion principles in Nominal Logic. Nominal Logic is originally a non-standard logic, parameterized by various categories of *atoms* (a.k.a. names, or variables); there are built-in notions of swapping (primitive) and freshness of an atom for an object (derived) and an underlying assumption of *finite support*, essentially saying that, for each object, all but a finite number of atoms are fresh for it; all the expressible predicates have a property called *equivariance* (invariance under swapping). This non-standard logic is discussed, e.g., in [49]. On the other hand, the nominal approach can also be developed in a standard logic such as classic HOL, as shown by the Isabelle/HOL Nominal Package

[153, 149, 148].

Next, we follow [116] and [148] to recall the Nominal approach in more technical terms. ([148] poses conditions slightly more general than [116], required for working with classical logic.) As usual, we consider the syntax of the untyped λ-calculus, **term**, based on the fixed set of variables (atoms) **var**. Let **perm**, ranged over by p, be the set of *permutations*, which are simply lists of pairs of variables. All the sets S involved in the following discussion are required to come equipped with a swapping operator $\_[\_]^K : K \to \mathbf{perm} \to K$, subject to some expected properties of permutation swapping, where, given $k \in K$, we think of $k[p]^K$ as k with all the pairs in p swapped in it consecutively. We shall write $k[p]$ instead of $k[p]^K$. If p is a singleton list $[(z_1, z_2)]$, we write $k[z_1 \wedge z_2]$ instead of $k[p]$. Standard set constructs have canonical ways to extend the swapping operator. For instance, given $f : K \to L$, $f[p] : K \to L$ is $\lambda k.\ f(k[p^{\mathsf{rev}}])[p]$, where p^{rev} is the reverse of p. By the properties assumed for swapping, $\_[z_1 \wedge z_2]$ and its reverse, $\_[z_2 \wedge z_1]$, are actually the same operator. A specific feature of Nominal is the *definability of freshness by swapping*. The *support* of an item $k \in K$, written $\mathsf{supp}\ k$, is the following subset of **var**: $\{x.\ \mathsf{finite}\ \{y.\ k[x \wedge y] \neq k\}\}$. "Support" is the Nominal terminology for "the set of free variables"; thus, "generalized freshness", $\mathsf{gFresh}\ x\ k$ (written $x \#\ k$ in Nominal jargon) is taken to mean $x \notin \mathsf{supp}\ k$.

The Nominal recursor employs the syntactic constructs Var, App and Lm, as well as the above notion of permutation swapping. All the operators involved in the recursive clauses in a presumptive definition with target domain A (assumed to have a notion of permutation swapping on it, as discussed above), in our notation $\mathsf{gVar} : \mathbf{var} \to A$, $\mathsf{gApp} : \mathbf{term} \to A \to \mathbf{term} \to A \to A$ and $\mathsf{gLm} : \mathbf{var} \to \mathbf{term} \to A \to A$, are required to have finite support. Moreover, the following *Freshness Condition for Binders* (FCB) is required to hold for all $x \in \mathbf{var}$, $X \in \mathbf{term}$ and $a \in A$:

- $\mathsf{gFresh}\ x\ \mathsf{gLm} \wedge \mathsf{finite}\ (\mathsf{supp}\ a) \Longrightarrow \mathsf{gFresh}\ x\ (\mathsf{gLm}\ x\ X\ a)$.

Under the above conditions, we obtain an operator $H : \mathbf{term} \to A$ which is "almost a morphism" w.r.t. the syntactic constructs, in that it commutes with Var and App in the usual sense and commutes with Lm for fresh names only, namely:

- $H\ (\mathsf{Lm}\ x\ X) = \mathsf{gLm}\ x\ X\ (H\ X)$ provided $\mathsf{gFresh}\ x\ \mathsf{gVar}$, $\mathsf{gFresh}\ x\ \mathsf{gApp}$ and $\mathsf{gFresh}\ x\ \mathsf{gLm}$ hold.

(Commuting for "fresh names only" is not a restriction of the above combinator, but rather a feature that matches standard practice in syntax with bindings – see Section 6 in [148].)

Compared to our main substitution-based recursion combinator (henceforth abbreviated SRC), the above Nominal recursion combinator (henceforth abbreviated NRC) has the advantage of uniformity and economy of structure (only swapping is primitive, and everything is based on it). Moreover, the consideration of the aforementioned "almost morphisms" as targets of definitions, w.r.t. which freshness for parameters may be assumed, provides extra generality to the NRC (this can also be made available in our setting, but we have not formalized it yet). On the other hand, the conditions that ARC requires to be checked by

the user are simpler than those required by NRC, the latter involving a certain quantifier complexity, notably when considering the support of functions (although, like hours, often these conditions can be checked automatically, as shown by the various examples considered by work using the Nominal Package). Moreover, there is some benefit in staying first-order as in our Horn approach, and not getting into the second-order issue of finiteness of support. For example, as soon as one deals with more semantic situations, such as our semantic-domain interpretation from Section 2.3.1, the finite support conditions are no longer satisfied, but a complex combination of induction and recursion is need to have a Nominal definition eventually go through, as shown in [116] (on page 492) for this very example. (On the other hand, it is true that as far as denotational-like semantics is concerned, any approach may incur difficulties, as was our case with the de Bruijn example from Section 2.6.2.) Another main difference between NRC and SRC is of course the presence in SRC of built-in compositionality, which relates to the user as both an obligation and a reward.

Michael Norrish's work. This is perhaps the approach most related to hours. [104] introduces a recursor for λ-calculus involving swapping and the free-variable operator FV, and also considers parameters a la Nominal. If we ignore the largely orthogonal extra generality of op. cit. w.r.t. parameters, and replace the use of $\mathsf{FV} : \mathbf{term} \to \mathbf{P}_f(\mathbf{var})$ with the (complementary) use of $\mathsf{fresh} : \mathbf{var} \to \mathbf{term} \to \mathbf{bool}$, we obtain something equivalent to the Horn theory consisting of the following:
- all the clauses from our swapping-based variation from Section 2.8.1, except for CSw;
- additionally, the following clauses (included in the predicate swapping defined in op. cit. on page 249):
— $X[z \wedge z] = X$,
— $X[x \wedge y][x \wedge y] = X$,
— $\mathsf{fresh}\ x\ X\ \wedge\ \mathsf{fresh}\ y\ X \Longrightarrow X[x \wedge y] = X$,
— $\mathsf{fresh}\ x\ (X[z_1 \wedge z_2]) = \mathsf{fresh}\ (x[z_1 \wedge z_2]^v)\ X$,
and the given recursor being the associated Horn recursor. Therefore, [104] did in fact propose a Horn-like recursive principle for syntax with bindings. Interestingly, op. cit. also contemplates considering substitution, but renounces in favor of swapping because "permutations move around terms much more readily than substitution" (op. cit., page 248). However, as we have shown in this chapter, the (indeed harder) task of having substitution "move around terms" results in a recursion principle which is *not* harder to apply (from the user's perspective), while bringing the significant reward of substitution compositionality.

Abstract internal characterization of terms with bindings. Such characterizations, similar to ours given in Th. 2.14, can be found in [58], [105] and [79].

The characterization from (the already classic paper) [58] employs essentially the follow-

ing:
- our clauses for freshness preservation and reflection, F1-F4 and F1^r-F4^r (Axiom 1 in op. cit.);
- our clauses for substitution, S1-S4, together with the clause $(\mathsf{Lm}\ y\ X)[Y/y] = \mathsf{Lm}\ y\ X$ (Axiom 2 in op. cit.);
- our clause AR (Axiom 3 in op. cit.);
- an auxiliary weak-HOAS-like operator $\mathsf{ABS} : (\mathbf{var} \to \mathbf{term}) \to \mathbf{term}$, together with a statement of its relationship with Lm via variable-for-variable substitution, namely: $\mathsf{ABS}\ (\lambda y.\ X[y/x]) = \mathsf{Lm}\ x\ X$ (Axiom 4 in op. cit.);
- the natural weak-HOAS-like iteration principle based on Var, App and ABS (Axiom 5 in op. cit.) – see also Prop. 3.4 in the next chapter.

The characterization from [105] employs the syntactic constructs, free variables and the notion of permutation from Nominal Logic.

The characterization from [79] achieves (a certain degree of) abstractness by the datatype mechanism of the underlying specification language (Maude [31]), even though it operates along the lines of the concrete recipe for quasi-terms sketched in Section 1.4.1. Namely, substitution is defined fully, as if on quasi-terms, employing an arbitrary pickFresh function to rename the bound variables while traversing bindings. However, equations defining α-equivalence, as if "after the fact", are actually interpreted by Maude as "working" in parallel with those for substitution, making all the involved equations operating on terms (not quasi-terms) as an abstract data type. The arbitrary choice behind pickFresh still accounts for the "concrete" nature of the specification though. (But a purely equational version of our Horn-based results, taking fresh as a Boolean-valued operation, and thus working two-sortedly essentially with freshness-reflecting models by default, is easily implementable in Maude.)

What distinguishes our characterization among the above ones is its minimality: no (permutation) swapping, pickFresh, or other auxiliaries. The only operators are the essential ones: the constructs, freshness and substitution. While the clauses that compose this characterization are of course well-known, the fact that these clauses are sufficient to characterize terms seems to be a new result. In fact, our very insistence on, and formalization of, uniqueness (up to isomorphism), as well as the consideration of parts of this uniqueness (in Theorems 2.11, 2.13 and 2.12) with the purpose of improving facts about the recursor seem to form a path so far unexplored in the world of theorem proving.

Non-standard-model approaches to syntax. These include the work based on functor categories from [46, 67] and the more elementary approach from [14] (the latter also motivated by some functor category mathematics). These approaches add more structure to terms, regarding them as *terms in contexts* – this is inspired by the method of *de Bruijn levels* [36], and consists conceptually in "turning the free-variable function $\mathsf{FV} : \mathbf{term} \to \mathbf{P}(\mathbf{var})$ into

extra structure on terms" ([46], page 3). Here, there is no single set of terms, but rather terms form a family of sets indexed by the "contexts", with additional categorical structure (i.e., as sheafs). The flexibility of moving between different contexts allows one to overcome the typical problems with recursive definitions on syntax.

[14] proposes a *weak HOAS* representation (modeling binders as functions between variables and terms, and then ruling out the so-called "exotic terms") in conjunction with considering terms in infinite contexts. The latter set, of terms (or expressions) in infinite contexts, denoted **eic**, is thus $(I\!N \to \mathbf{var}) \to \mathbf{exp}$, where **exp** is the set of weak HOAS terms. The outcome is an iterator which, given any target set A, has type $(\mathbf{var} \to A) \to (A \to A \to A) \to (A \to A) \to \mathbf{eic} \to A$, with $\mathbf{var} \to A$, $A \to A \to A$ and $A \to A$ above corresponding to the recursive cases of variable injection, application and λ-abstraction, respectively. The type of this iterator is precisely the one for de Bruijn terms (with λ-abstraction a unary operator, not introducing any binders). However, because of the more involved type **eic**, the treatment of terms offers flexibility for dealing properly with standard bindings. Interestingly, **eic** is essentially the type of the "semantic" interpretation toDB of λ-terms as de Bruijn terms that we discussed in Section 2.9.4, and the equations for the above iterator (Th. 2 on page 6 in op. cit.) are essentially referring to the image of toDB – this constitutes an alternative explanation for the FOAS type of this weak-HOAS based iterator.

[67] and [46] take (independently) very similar approaches. Let F_ω be the category of natural numbers, regarded as sets, and functions between them. Each number $n = \{0, \ldots, n-1\}$ is to be thought as the context consisting of the first n variables (given a fixed ordering on **var**, i.e., a bijection $(x_i)_{i \in I\!N}$), $\{x_0, \ldots, x_{n-1}\}$. A presheaf is a functor from F_ω to **Set**, the category of sets. Let **Sh** be the category of presheaves and natural transformations. This category has products and sums, taken componentwise, and is cartesian-closed. Let us denote by $\oplus$, $\otimes$ and $\Rightarrow$ the sum, the product and its adjoint, respectively, in **Sh**.

The variables are organized as a presheaf **Var** by taking:
- For each n, $\mathbf{Var}_n$ to be $\{x_0, \ldots, x_{n-1}\}$;
- For each $f : m \to n$ and $x_i \in \mathbf{Var}_m$, $(\mathsf{Var}_n\ f)\ x_i$ to be $x_{f\ i}$.

Moreover, the λ-calculus terms are organized as a presheaf **Term** by taking:
- For each n, $\mathbf{Term}_n$ to be $\{X \in \mathbf{term}.\ \forall m \geq n.\ \mathsf{fresh}\ x_m\ X\}$;
- For each $f : m \to n$ and $X \in \mathbf{Term}_m$, $(\mathbf{Term}_n\ f)\ X$ to be the term obtained from X by substituting each x_i with $x_{f\ i}$.

An interesting feature of $\Rightarrow$ is that, for any sheaf K, $\mathbf{Var} \Rightarrow K$ is isomorphic to $\delta\ K$, the *shift* (or *context extension*) of K, defined as follows:
- For each n, $(\delta\ K)\ n = K\ (n+1)$;
- For each $f : m \to n$ and $k \in (\delta\ K)\ n$, $(\delta\ K)\ f\ k = K\ (\lambda\ i < m+1.\ \mathsf{if}\ i < m\ \mathsf{then}\ f\ i\ \mathsf{else}\ \mathrm{n{+}1})\ k$.

In what follows (more specifically, in clause (3) below), we assume the isomorphism is an

equality, i.e., $(\mathbf{Var} \Rightarrow K) = \delta\ K$.

Now, we define the **Sh**-endofunctor T by $T = \lambda K.\ \mathbf{Var} \oplus (K \otimes K) \oplus (\mathbf{Var} \Rightarrow K)$ and the following natural transformations, organizing **Term** as a T-algebra:
- $\mathsf{VAR} = (\mathsf{VAR}_n)_{n \in I\!N} : \mathbf{Var} \to \mathbf{Term}$, by $\mathsf{VAR}_n\ x = \mathsf{Var}\ x$;
- $\mathsf{APP} = (\mathsf{APP}_n : (\mathbf{Term} \otimes \mathbf{Term})_n = \mathbf{Term}_n \times \mathbf{Term}_n \to \mathbf{Term}_n)_{n \in I\!N}$,
by $\mathsf{APP}_n\ X\ Y = \mathsf{App}\ X\ Y$;
- $\mathsf{LM} = (\mathsf{LM}_n : (\mathbf{Var} \Rightarrow \mathbf{Term})_n = (\delta\ \mathbf{Term})_n = \mathbf{Term}_{n+1} \to \mathbf{Term})_{n \in I\!N}$,
by $\mathsf{LM}_n\ X = \mathsf{Lm}\ x_{n+1}\ X$.
(Notice again the weak-HOAS-like type for the λ-case.)

It turns out that that **Term** is the initial T-algebra, which yields iteration and recursion principles in the category **Sh**.

Compared to the more elementary settings discussed previously (which include our own), the non-standard model approach has the advantage of "syntactic purity" (as only the term constructs are involved) and mathematical elegance and generality,[31] but does have a couple of disadvantages too. Thus, it is harder to formalize in a standard theorem prover, due to the fancy category theory involved (although this is more of a (solvable) problem for the implementor, not for the user). More importantly, it seems rather tedious to employ in order to define concrete operators. Indeed, the target domain would have to be organized each time as an algebra for the above functor T, which means that a lot of structure and verifications would need to be provided and performed by the user. On the other hand, a category theorist may argue that all this effort would not go in vain, as the extra structure will be helpful later in proofs – and this is much like our argument in favor of "built-in" substitution made in this chapter. Concerning the latter, both [67] and [46] show how to define substitution using the above iterator. Moreover, [46] goes further and integrates substitution in the algebraic structure, inferring a version of recursion with a built-in substitution lemma, which is essentially what we do in this chapter in a more standard setting. Interestingly, it is shown that incorporating unary substitution is essentially the same as incorporating parallel substitution (the resulted categories, of *monoids* and *clones*, are equivalent), corresponding to the intuition that parallel substitution brings nothing essentially new beyond unary substitution – this type of mathematical insight obtained by climbing to a more abstract level can also be counted as an advantage of categorical approaches.

Explicit substitution. In a way, our substitution-based recursion pushes an agenda, similar to that of the *$\lambda\sigma$-calculus* (with explicit substitution) from [8], of considering substitution a first-class citizen. However, op. cit. focusses on more (operational-) semantic issues like reduction confluence and termination, while we focus on syntax.

[31] However, note that, although more abstract, it certainly does not capture the discussed elementary approaches as particular cases - the latter employ the standard topos of sets, while the former employ a non-standard sheaf topos tailored to handle bindings uniformly.

Efficient representation of terms. This subject is not strictly related to our contribution. As mentioned, our view is behavioral, meaning we care less about the route than about the end-product. However, any development of very large size such as ours needs to take the issue of efficiency into account at some point, especially if it keeps on growing with new facts and features (as does ours). Our formalization based on α-classes is faithful to the standard theory, but is perhaps less efficient than one of the following approaches:
- a de Bruijn-based one;
- one that tries to make the best of de Bruijn while still retaining "named" variables, such as [43] or [139, 118];
- one based on partial functions, as in the Nominal Package [148, 149].
In the future, we may rewrite our underlying implementation of terms using one of these approaches.

Formalized examples and case studies. Norrish has formalized his swapping-based recursion principle for the syntax of untyped λ-calculus and has included it in the distribution of the HOL4 system [7].

The Nominal package [151] has been used in several formal developments, including proofs of Church-Rosser and Standardization for λ-calculus – the classic latter result, presented, e.g., in Barendregt, is different than the call-by-name and call-by-value standardization theorems from [117] that we have formalized, in that the latter deals with more programming-oriented strategies, not allowing rewrites under λ-abstractions.

In spite of the extensive (and growing) interest in HOAS as a formal means of specification and verification and in its associated adequacy proofs, our adequacy proof discussed in Sections 2.3.2, 2.5.2 and 2.9.4 seems to be the first reported in the literature.[32] Even though it is performed for a fairly simple (albeit standard) logical framework, λ-calculus with constants, the actual proof technique works for more complex frameworks such as LF too. Some of the meta-theory of LF has been formalized using the Nominal package, but we are not aware of actual representations and adequacy proofs performed in this setting.

[32] Here we refer to the traditional, LF-style HOAS encodings and adequacy. We also have a formalized notion of adequacy in Chapter 3, but there we change the traditional roles for the actors: object system, logical framework and meta-logical framework.

Chapter 3

HOAS [1]

3.1 Introduction

In this chapter, we advocate a variant of the HOAS approach (II.b) described in Section 1.2.2, namely, the general-purpose framework approach, highlighting an important feature, apparently not previously explored in the HOAS literature: the capability to *internalize, and eventually automate, both the representation map and the adequacy proof.*

Let us illustrate this point by an example. Say we wish to represent and reason about λ-calculus and its associated β-reduction (as we actually do in this chapter). Therefore, the object system is a "standalone", Platonic mathematical notion, given by a collection of items called λ-terms, with operators on them, among which the syntactic constructs, free variables and substitution, and with an inductively defined reduction relation – typically, these are initially (informally) FOAS-specified, as in Chapter 2.

In the HOAS-tailored framework approach, for representing this system one defines a corresponding collection of constants in the considered logical framework, say LF, and then does an informal (but rigorous) *pen and paper proof* of the fact that the syntax representation is adequate (i.e., the existence of a compositional bijection between the λ-terms and normal forms of LF terms of appropriate types) and of a corresponding fact for β-reduction [109].

In the general-purpose framework approach, one can define the original system itself (here λ-calculus) in the meta-logical framework (say, HOL_ω, Higher-Order Logic with Infinity) *in such a way that accepting this definition as conforming to the mathematical definition is usually not a problem* (for one who already accepts that HOL_ω is adequate for representing the mathematical universe (or part of it)), since the former definitions are typically almost verbatim renderings of the latter – in HOL_ω, one can define inductively the datatype of terms, perhaps define α-equivalence and factor to it, then define substitution, reduction, etc. Moreover, one can also define in HOL_ω a system that is a HOAS-style representation of (the original) λ-calculus, i.e.: define a new type of items, call them HOAS-terms, with operators corresponding to the syntactic constructs of the original terms, but dealing with bindings via higher-order operators instead. In particular, the constructor for λ-abstraction will

have type (HOAS-terms $\not\to$ HOAS-terms) $\to$ HOAS-terms, where one may choose the type constructor $\not\to$ to yield a restricted function space, or the whole function space accompanied by a predicate to cut down the "junk", etc. Once these constructions are done, one may also define in HOL_ω the syntax representation map from λ-terms to HOAS-terms and prove adequacy. (And a corresponding effort yields the representation of λ-term reduction.) Now, if the above are performed in a theorem prover that implements HOL_ω, such as Isabelle/HOL, then HOAS-terms become a *formally certified* adequate representation of the (original) λ-terms, not available in the existent HOAS-tailored approaches. Moreover, in many cases the construction of the HOAS-terms, the proofs of their basic properties and the adequacy proof can be *automated*, being essentially the same for all syntaxes with bindings.

One may argue that, on the other hand, the above HOAS-terms do not retain all the convenience of a genuine HOAS encoding. Thus, when various standard relations need to be defined on HOAS-terms, certain context-free clauses specific to the HOAS-tailored frameworks (within the so-called *HOAS-encoding of judgements*) are no longer available here. E.g., a rule like

$$\frac{\forall X.\ X : S \Longrightarrow A\,X : T}{\mathsf{Lam}(A) : S \to T}$$

(typing rule for λ-abstractions – $\Longrightarrow$ is logical implication and A a map from terms to terms) cannot act as a definitional clause in HOL_ω for a typing relation $\_:\_$, due to non-monotonicity. The short answer to this objection is agreeing that general-purpose frameworks do bring their expressiveness with the price of not allowing the cleanest possible HOAS. A longer answer is given in this chapter, where we argue that developing suitable general-purpose-framework concepts accommodates non-monotonicity and impredicativity flavors that make "pure" HOAS so attractive.

We develop HOAS concepts and techniques pertaining to the general-purpose framework approach. Here, the general-purpose framework could be regarded as being the mathematical universe (given axiomatically by any standard formalization of mathematics). All the involved systems, including the original systems and their representations, dwell in this mathematical universe, and are thus discussed and related via standard-mathematics concepts and theorems. Our HOAS case study is a proof of the strong normalization result for System F (a.k.a. the *polymorphic second-order λ-calculus*) [53]. In the context of the (general-purpose) HOAS concepts introduced in this chapter, the statement of the strong normalization problem and the natural attempt to start proving it brings one quickly into the heart of the problem and, to a degree, suggests its resolution.

Apart from this introduction, Section 3.2 recalling some basic facts about reduction and typing, Section 3.6 giving some pointers to our Isabelle formal scripts, and Section 3.7 discussing related work, this chapter has two main parts. In the first part, consisting of Sections 3.3 and 3.4, we discuss some general HOAS techniques for representing and

reasoning about syntax and inductively defined relations (i.e., a HOAS take on main themes of Chapter 2), illustrated on the λ-calculus and System F. The HOAS "representation" of the original first-order syntax will not be a representation in the usual sense (via defining a new (higher-order) syntax), but will take *a different view of the same syntax*. Recall from the beginning of Section 2.9.1 that abstractions have the form $\mathsf{Abs}\ x\ X$, i.e., are essentially pairs (x, X) variable-term modulo α-equivalence, and of that the Lam-operator takes abstractions as arguments. Under the higher-order view, abstractions A are no longer constructed by variable-term representatives, but are analyzed/"destructed" by applying them (as functions), via substitution, to terms. Namely, given a term X, A_X, read "A applied to X", is defined to be $Y[X/x]$, where (x, Y) is any variable-term representative for A. This way, the collection of abstractions becomes essentially a restricted function space from terms to terms, as in strong HOAS. (The [strong HOAS]–[weak HOAS] dichotomy is revisited in Section 3.7.) Although this change of view is as banal as possible, it meets its purpose: the role previously played by substitution now belongs to function-like application. The latter of course originates in substitution, but one can forget about its origin. In fact, one can (although is not required to!) also forget about the original first-order binding constructor and handle terms entirely by means of the new, higher-order destructor. Moving on to the discussion of recursive-definition principles for syntax, we perform an analysis of various candidates for the type of the recursive combinator, resulting notably in a novel "impredicative" principle in the spirit of (strong) HOAS.

Then we discuss HOAS representation of inductively defined relations, performed by a form of transliteration following some general patterns. These patterns are illustrated by the case of the reduction and typing relation for System F, and it appears that a large class of systems (e.g., most of those from the monographs [18, 60, 90, 114]) can be handled along these lines. For typing, we also present a "purely HOAS" induction principle, not mentioning typing contexts. Once our formalization will be fully automated (see Section 5.2), it will have a salient advantage over previous HOAS approaches: adequacy will need *not* be proved by hand, but will follow automatically from general principles.

In the second part, Section 3.5, we sketch a proof of strong normalization for System F within our HOAS framework. We make essential use of our aforementioned definitional principle and typing-context-free induction principle to obtain a general criterion for proving properties on typable terms (which is in principle applicable to properties other than strong normalization, including confluence and type preservation). Unlike previous proofs [54, 144, 91, 51, 19, 12, 22, 75, 41], our proof does not employ *data or type environments* and *semantic interpretation of typing contexts* – a virtue of our setting, which is thus delivering the HOAS-prescribed service of *clearing the picture of inessential details*.

3.2 The λ-calculus reduction and the System F typing system recalled

The two systems, β-reduction for λ-calculus and the typing system for System F, are standardly defined employing FOAS. We later refer to them as "the original systems", to contrast them with their HOAS representations.

3.2.1 The (untyped) λ-calculus under β-reduction

We shall consider the abstraction-based variant of the λ-calculus syntax which we described at the beginning of Section 2.9.1. Thus, we fix set **var**, of *variables*, ranged over by x, y, z. The sets **term**, of *terms*, ranged over by X, Y, Z, and **abs**, of *abstractions*, ranged over by A, B, are given by:

$$\begin{array}{lcl} X & ::= & \mathsf{Var}\ x \mid \mathsf{App}\ X\ Y \mid \mathsf{Lam}\ A \\ A & ::= & \mathsf{Abs}\ x\ X \end{array}$$

where we assume that, within $\mathsf{Abs}\ x\ X$, x is bound in X (and terms and abstractions are identified modulo α-equivalence). For brevity, we shall write $x.X$ instead of $\mathsf{Abs}\ x\ X$. (Notice that, in Section 2.9.1, we wrote $\mathsf{Lm}\ x\ X$ as a shorthand for $\mathsf{Lam}(\mathsf{Abs}\ x\ X)$, i.e., for what in this chapter shall be written as $\mathsf{Lam}(x.X)$ – the latter notation is more convenient for the abstraction-oriented approach from this chapter.)

Also, recall our habit to keep implicit the injective map $\mathsf{Var} : \mathbf{var} \to \mathbf{term}$, and pretend that $\mathbf{var} \subseteq \mathbf{term}$ (this omission will be performed directly for the syntax of System F below).

Recall that we write:

- $\mathsf{fresh} : \mathbf{var} \to \mathbf{term} \to \mathbf{bool}$, for the predicate indicating if a variable is fresh in a term;
- $\_[\_] : \mathbf{term} \to \mathbf{env} \to \mathbf{term}$, for the parallel substitution, where **env**, the set of *environments*, ranged over by ρ, consists of partial functions of finite domain from **var** to **term** (thus, $X[\rho]$ is the term obtained from X by (capture-free) substituting each of its free variables x with $\rho\ x$ if the latter is defined);
- $\_[\_/\_] : \mathbf{term} \to \mathbf{term} \to \mathbf{var} \to \mathbf{term}$, for substitution.

We employ the same notations for abstractions: $\mathsf{fresh} : \mathbf{var} \to \mathbf{abs} \to \mathbf{bool}$, $\_[\_] : \mathbf{abs} \to \mathbf{env} \to \mathbf{abs}$, etc.

The *one-step β-reduction* $\rightsquigarrow : \mathbf{term} \to \mathbf{term} \to \mathbf{bool}$ is defined inductively by the following rules:

$$\frac{\cdot}{\mathsf{App}\ (\mathsf{Lam}(x.Y))\ X \rightsquigarrow Y[X/x]}(\mathsf{Beta}) \qquad \frac{X \rightsquigarrow Y}{\mathsf{Lam}(z.X) \rightsquigarrow \mathsf{Lam}(z.Y)}(\mathsf{Xi})$$

$$\frac{X \rightsquigarrow Y}{\mathsf{App}\ X\ Z \rightsquigarrow \mathsf{App}\ Y\ Z}(\mathsf{AppL}) \qquad \frac{X \rightsquigarrow Y}{\mathsf{App}\ Z\ X \rightsquigarrow \mathsf{App}\ Z\ Y}(\mathsf{AppR})$$

X is called *strongly normalizing* if there is no infinite sequence $(X_n)_{n \in \mathbb{N}}$ with $X_0 = X$

and $\forall n.\ X_n \rightsquigarrow X_{n+1}$.

3.2.2 System F

It was introduced independently by Jean-Yves Girard in [53] in the context of proof theory and by John Reynolds in [130] for the study of programming language polymorphism. We describe this system as a typing system for λ-terms without type annotations, in a Curry style (see [18]).

Its syntax consists of two copies of the untyped λ-calculus syntax – one for data and one for types. (Of course, β-reduction will only be meaningful for data.) More precisely, we fix two infinite sets, **dvar**, of *data variables* (*dvars* for short), ranged over by x, y, z, and **tvar**, of *type variables* (*tvars* for short), ranged over by tx, ty, tz. The sets **dterm** and **dabs**, of *data terms* and *abstractions* (*dterms* and *dabstractions* for short), ranged over by X, Y, Z and A, B, C, and **tterm** and **tabs**, of *type terms* and *abstractions* (*tterms* and *tabstractions* for short), ranged over by tX, tY, tZ and tA, tB, tC, are defined by the following grammar, again up to α-equivalence:

$$\begin{array}{lcl} X & ::= & x \mid \mathsf{App}\ X\ Y \mid \mathsf{Lam}\ A \\ A & ::= & x.X \\ tX & ::= & tx \mid \mathsf{Arr}\ tX\ tY \mid \mathsf{Al}\ tA \\ tA & ::= & tx.tX \end{array}$$

Above, App and Lam stand, as in Section 3.2.1, for "application" and "lambda", while Arr and Al stand for "arrow" and the "for all" quantifier (also interpreted as product). Since dterms do not have type annotations, indeed both the abstract syntax of dterms and that of tterms are that of λ-calculus (from Section 3.2.1), just that for tterms we write Arr and Al instead of App and Lam.

All concepts from Section 3.2.1 apply to either syntactic category, separately. Let **denv**, ranged over by ρ, be the set of data environments, and **tenv**, ranged over by ξ, that of type environments. For any items a and b, we may write $a : b$ for the pair (a, b). A *well-formed typing context* (*context* for short) $\Gamma \in$ **ctxt** is a list of pairs dvar-tterm, $x_1 : tX_1, \ldots, x_n : tX_n$, with the x_i's distinct. The homonymous predicates $\mathsf{fresh} : \mathbf{dvar} \to \mathbf{ctxt} \to \mathbf{bool}$ and $\mathsf{fresh} : \mathbf{tvar} \to \mathbf{ctxt} \to \mathbf{bool}$ (indicating if a dvar or a tvar is fresh for a context) are defined as expected:

- $\mathsf{fresh}\ y\ [] = \mathsf{True}$;
- $\mathsf{fresh}\ y\ (\Gamma, (x : tX)) = (\mathsf{fresh}\ y\ \Gamma \wedge y \neq x)$;
- $\mathsf{fresh}\ ty\ [] = \mathsf{True}$;
- $\mathsf{fresh}\ ty\ (\Gamma, (x : tX)) = (\mathsf{fresh}\ ty\ \Gamma \wedge \mathsf{fresh}\ ty\ tX)$.

The type inference relation $(\_ \vdash \_ : \_) : \mathbf{ctxt} \to \mathbf{dterm} \to \mathbf{tterm} \to \mathbf{bool}$ is defined

inductively by the clauses:

$$\frac{\cdot}{\Gamma, x : tx \vdash x : tx}\ (\mathsf{Asm})\ [\mathsf{fresh}\ x\ \Gamma] \qquad \frac{\Gamma \vdash X : tx}{\Gamma, y : tY \vdash X : tx}\ (\mathsf{Weak})\ [\mathsf{fresh}\ y\ \Gamma]$$

$$\frac{\Gamma, x : tx \vdash Y : tY}{\Gamma \vdash \mathsf{Lam}(x.Y) : \mathsf{Arr}\ tx\ tY}\ (\mathsf{ArrI})\ [\mathsf{fresh}\ x\ \Gamma] \qquad \frac{\Gamma \vdash Y : tY}{\Gamma \vdash Y : \mathsf{Al}(tx.tY)}\ (\mathsf{AlI})\ [\mathsf{fresh}\ tx\ \Gamma]$$

$$\frac{\Gamma \vdash X : \mathsf{Arr}\ tY\ tZ \quad \Gamma \vdash Y : tY}{\Gamma \vdash \mathsf{App}\ X\ Y : tZ}(\mathsf{ArrE}) \qquad \frac{\Gamma \vdash Y : \mathsf{Al}(tx.tY)}{\Gamma \vdash Y : tY[tx/tx]}(\mathsf{AlE})$$

We write $\vdash X : tX$ for $[] \vdash X : tX$. A term X is called *typable* if $\Gamma \vdash X : tX$ for some Γ and tX.

3.3 HOAS view of syntax

Here we present a HOAS approach to the *syntax* of calculi with bindings. We describe our approach for the paradigmatic particular case of the untyped λ-calculus, but our discussion is easily generalizable to terms generated from any binding signature (as in Section 2.9.1). We do *not* define a new higher-order syntax, but introduce higher-order operators on the original syntax – hence we speak of a *HOAS view* rather than of a *HOAS representation*.

3.3.1 Abstractions as functions

Throughout the rest of this section, we use the concepts and notations from Section 3.2.1, and *not* the ones from Section 3.2.2. Given $A \in \mathbf{abs}$ and $X \in \mathbf{term}$, the *functional application of A to X*, written A_X, is defined to be $Y[X/x]$ for any x and Y such that $A = (x.Y)$. (The choice of (x, Y) is easily seen to be immaterial.) The operator $\_$ is extensional, qualifying the set of abstractions as a *restricted term-to-term function space*, and preserves freshness. Thus, abstractions are no longer regarded as pairs var-term up to α-equivalence, but as functions, in the style of HOAS. Under this higher-order view, abstractions can be destructed by application, as opposed to constructed by means of var-term representatives as in the original first-order view. But does the higher-order view suffice for the specification of relevant systems with bindings? I.e., can we do without "constructing" abstractions? Our answer is threefold:

- (1) Since the higher-order view does not change the first-order syntax, abstractions by representatives are still available if needed.
- (2) Many relevant systems with bindings employ the binding constructors within a particular style of interaction with substitution and scope extrusion (e.g., all variables appear either bound, or substituted, or [free in the hypothesis]) which makes the choice of binding representatives irrelevant. This phenomenon, to our knowledge not yet rigorously studied

mathematically for a general syntax with bindings, is really the basis of most HOAS representations from the literature. In Section 3.4, we elaborate informally on what this phenomenon becomes in our setting.

- (3) The previous point argued that relevant systems *specifications* can do without constructing abstractions. Now, w.r.t. *proofs* of meta-theoretic properties, one may occasionally need to perform case-analysis and induction *on abstractions*. HOAS-style case-analysis and induction are discussed below, after we introduce 2-abstractions.

3.3.2 2-abstractions

These are for abstractions what abstractions are for terms. 2-abstractions $\mathcal{A} \in \mathbf{abs2}$ are defined as pairs $x.A$ variable-abstraction up to α-equivalence (just like abstractions are pairs variable-term up to α). (Alternatively, they can be regarded as triples $x.y.Z$, with $x, y \in \mathbf{var}$ and $Z \in \mathbf{term}$, again up to α.) Next we define two application operators for 2-abstractions. If $\mathcal{A} \in \mathbf{abs2}$ and $X \in \mathbf{term}$, then $\mathcal{A}\_1\, X$ and $\mathcal{A}\_2\, X$ are the following elements of $\mathbf{abs}$:

- $\mathcal{A}\_1\, X = A[X/x]$, where x, A are such that $\mathcal{A} = (x.A)$;
- $\mathcal{A}\_2\, X = (y.(Z[X/x]))$, where y, Z are such that $y \neq x$, $\mathsf{fresh}\ y\ X$ and $\mathcal{A} = (y.(x.Z))$.

(Again, the choice of representatives is immaterial.) Thus, essentially, 2-abstractions are regarded as 2-argument functions and applied correspondingly.

Now we can define homonymous syntactic operations for abstractions lifting those for terms:

- $\mathsf{Var} : \mathbf{var} \to \mathbf{abs}$, by $\mathsf{Var}\ x = (y.x)$, where y is such that $y \neq x$;
- $\mathsf{App} : \mathbf{abs} \to \mathbf{abs} \to \mathbf{abs}$, by $\mathsf{App}\ A\ B = (z.\,(\mathsf{App}\ X\ Y))$, where z, X, Y are such that $A = (z.X)$ and $B = (z.Y)$.
- $\mathsf{Lam} : \mathbf{abs2} \to \mathbf{abs}$, by $\mathsf{Lam}\ \mathcal{A} = (x.\,(\mathsf{Lam}\ A))$, where x, A are such that $\mathcal{A} = (x.A)$.

(The definitions are correct, in that the choice of representatives is possible[2] and irrelevant.)

If we also define $\mathsf{id} \in \mathbf{abs}$ to be $(x.x)$ for some x, we can case-analyze abstractions by the above four (complete and non-overlapping) constructors:

Prop 3.1 *Given an abstraction A, one and only one of the following holds:*

- $A = \mathsf{id}$;
- $\exists x.\ A = \mathsf{Var}\ x$;
- $\exists B, C.\ A = \mathsf{App}\ B\ C$;
- $\exists \mathcal{A}.\ A = \mathsf{Lam}\ \mathcal{A}$.

Proof sketch. By simple verification. □

Functional application satisfies the expected exchange law,

[2]The issue of possibility is only raised for the App-case, where the representatives for A and B need to be synchronized on z.

- $(\mathcal{A} \_1\, X) \_\, Y = (\mathcal{A} \_2\, Y) \_\, X$,

and commutes with abstraction versus terms constructors (below, on the left we have the abstraction constructs, and on the right the term constructs):

- $(\mathsf{Var}\ x) \_\, X = x$,
- $(\mathsf{App}\ A\ B) \_\, X = \mathsf{App}\ (A \_\, X)\ (B \_\, X)$,
- $(\mathsf{Lam}\ \mathcal{A}) \_\, X = \mathsf{Lam}(\mathcal{A} \_1\, X)$.

Moreover, it commutes with parallel substitution,

- $(A \_\, X)[\rho] = ((A[\rho]) \_\, (X[\rho]))$,

and in particular with substitution:

- $(A \_\, X)[Y/y] = ((A[Y/y]) \_\, (X[Y/y]))$.

Notice that substitution (of a term for an explicitly indicated variable) is a first-order feature, abandoned when switching to the HOAS view. However, the above commutativity suggests that the FOAS and HOAS layers may be combined smoothly in proofs.

3.3.3 Induction principles for syntax

The following is the natural principle for terms under the HOAS view. Notice that it requires the use of abstractions.

Prop 3.2 *Let* $\varphi : \mathbf{term} \to \mathbf{bool}$ *be such that the following hold:*

- $\forall x.\ \varphi\ x$*;*
- $\forall X, Y.\ \varphi\ X \wedge \varphi\ Y \Longrightarrow \varphi(\mathsf{App}\ X\ Y)$*;*
- $\forall A.\ (\forall x. \varphi(A \_\, x)) \implies \varphi(\mathsf{Lam}\ A)$.

Then $\forall X.\ \varphi\ X$.

Proof sketch. By easy induction on the depth of X. $\square$

Likewise, a HOAS induction principle for abstractions requires the use of 2-abstractions. The 2-place application in the inductive hypothesis for Lam in Prop. 3.3 offers "permutative" flexibility for when reasoning about multiple bindings – the proof of Prop. 3.13 from Section 3.5 illustrates this.

Prop 3.3 *Let* $\varphi : \mathbf{abs} \to \mathbf{bool}$ *be such that the following hold:*

- $\varphi\ \mathsf{id}$*;*
- $\forall x.\ \varphi(\mathsf{Var}\ x)$*;*
- $\forall A, B.\ \varphi\ A \wedge \varphi\ B \Longrightarrow \varphi(\mathsf{App}\ A\ B)$*;*
- $\forall \mathcal{A}.\ (\forall x.\ \varphi(\mathcal{A} \_1\, x) \wedge \varphi(\mathcal{A} \_2\, x)) \Longrightarrow \varphi(\mathsf{Lam}\ \mathcal{A})$.

Then $\forall A.\ \varphi\ A$.

Proof sketch. By easy induction on the depth of A. $\square$

3.3.4 Recursive definition principles for syntax

This is known as a delicate matter in HOAS. One would like that, given any set $\mathcal{C}$, a map $H : \mathbf{term} \to \mathcal{C}$ be determined by a choice of the operations $\mathsf{cInV} : \mathbf{var} \to \mathcal{C}$, $\mathsf{cApp} : \mathcal{C} \to \mathcal{C} \to \mathcal{C}$, and cLam (whose type we do not yet specify) via the conditions:
(I) $H\,x = \mathsf{cInV}\;x$.
(II) $H(\mathsf{App}\,X\,Y) = \mathsf{cApp}\,(H\,X)\,(H\,Y)$.
(III) An equation (depending on the type of cLam) with $H(\mathsf{Lam}\;A)$ on the left.
(Here, we only discuss iteration, not full recursion.)

Candidates for the type of the operator cLam are:

(1) $\mathsf{cLam} : (\mathbf{term} \to \mathcal{C}) \to \mathcal{C}$, suggesting the equation $H(\mathsf{Lam}\;A) = \mathsf{cLam}(\lambda X.\, H(A\,\_\,X))$ – this is problematic as a definitional clause, due to its impredicativity;[3]

(2) A weak-HOAS-like [38, 58] variable-restriction of (1), namely, $\mathsf{cLam} : (\mathbf{var} \to \mathcal{C}) \to \mathcal{C}$, yielding the equation
(III$_w$): $H(\mathsf{Lam}\;A) = \mathsf{cLam}(\lambda x.\, H(A\,\_\,x))$
and a recursive principle:

Prop 3.4 *There exists a unique map $H : \mathbf{term} \to \mathcal{C}$ such that equations (I), (II), and (III$_w$) hold.*

Proof sketch.
Existence: Let **whTerm**, the set of "weak-HOAS terms", ranged over by P, Q, be given by the following grammar, where f denotes elements of $\mathbf{var} \to \mathbf{whTerm}$:

$$P ::= \mathsf{whVar}\;x \mid \mathsf{whApp}\;P\;Q \mid \mathsf{whLam}\;f$$

Then there is an embedding of **term** into **whTerm**, which commutes with the constructs. Our desired map H is the composition between the map $\mathbf{whTerm} \to \mathcal{C}$ obtained by standard iteration and this embedding.
Uniqueness: By easy induction on **term**. □

(3) $\mathsf{cLam} : (\mathcal{C} \to \mathcal{C}) \to \mathcal{C}$. Then there is no apparent way of defining the equation (III) in terms of Lam and cLam without parameterizing by valuations/environments in $\mathbf{var} \to \mathcal{C}$, and thus getting into first-order "details" (at least not in a standard setting such as ours – but see [141, 40] for a solution within a modal typed λ-calculus).

[3] Here, we call a definitional clause of a function "impredicative" if it is not guaranteed to avoid self-reference. Standard (set-theoretic, not domain-theoretic!) recursive definitions are predicative because for them there exists a well-founded relation w.r.t. which the arguments for their recursive calls are smaller.

(4) A "flattened" version (collapsing some type information) of both (1) and (3), namely, $\mathsf{cLam} : \mathbf{P}_{\neq\emptyset}(\mathcal{C}) \to \mathcal{C}$. This may be regarded as obtained by requiring the operator from (1) or (3) to depend only on the image of its arguments in $\mathbf{term} \to \mathcal{C}$ or $\mathcal{C} \to \mathcal{C}$, respectively. The natural associated (valuation-independent) condition (III) would be
- $H(\mathsf{Lam}\ A) = \mathsf{cLam}(\{H(A\_X).\ X \in \mathbf{term}\})$.

Unfortunately, this condition is still too strong to guarantee the existence of H. But interestingly, if we have enough variables, the existence of a compositional map holds:

Theorem 3.5 *Assume* $\mathsf{card}(\mathbf{var}) \geq \mathsf{card}(\mathcal{C})$ *and let* $\mathsf{cApp} : \mathcal{C} \to \mathcal{C} \to \mathcal{C}$ *and* $\mathsf{cLam} : \mathbf{P}_{\neq\emptyset}(\mathcal{C}) \to \mathcal{C}$ *(where* card *is the cardinal operator). Then there exists* $H : \mathbf{term} \to \mathcal{C}$ *such that:*
(I) $H(\mathsf{App}\ X\ Y) = \mathsf{cApp}\ (H\ X)\ (H\ Y)$ *for all* X, Y.
(II) $H(\mathsf{Lam}\ A) = \mathsf{cLam}(\{H(A\_X).\ X \in \mathbf{term}\})$ *for all* A.

Proof sketch. We employ the notion of interpretation in a semantic domain (from Section 2.3.1) for the domain C with operators $\mathsf{APP} = \mathsf{cApp}$ and $\mathsf{LM} : (C \to C) \to C$ defined by $\mathsf{LM}\ f = \mathsf{cLam}\ \{f\ c.\ c \in C\}$, obtaining a map $\_[\_] : \mathbf{term} \to \mathbf{val} \to \mathcal{C}$ (where $\mathbf{val} = (\mathbf{var} \to C)$) such that:
- (a) $[\mathsf{App}\ X\ Y]\ \rho = \mathsf{cApp}\ ([X]\,\rho)\ ([Y]\,\rho)$;
- (b) $[\mathsf{Lam}\,(y.X)]\ \rho = \mathsf{cLam}\ \{[X]\ (\rho[y \leftarrow c]).\ c \in C\}$.

Let now $\rho \in \mathbf{val}$ be a surjective valuation, whose existence is ensured by our cardinality hypothesis. We have the following:
- (c) $\{[X]\ (\rho[y \leftarrow c]).\ c \in C\} = \{[X[Y/y]]\ \rho.\ Y \in \mathbf{term}\}$.

Indeed, let L and R be the lefthand and righthand sides of the desired equation, and let M be $\{[X]\ (\rho[y \leftarrow \rho\ Y]).\ Y \in \mathbf{term}\}$. By the substitution lemma, we have $M = R$. Moreover, we have the following chain of equalities and inclusions:
$L = \{[X]\ (\rho[y \leftarrow \rho\ x]).\ x \in \mathbf{var}\}$ (by the surjectiveness of ρ)
$\subseteq M$ (since $\mathbf{var} \subseteq \mathbf{term}$)
$\subseteq L$ (by properties of sets).
Ultimately, $L = R$, as desired.

We define H by $H\ X = [X]\ \rho$. Then:
- (I) follows from (a);
- (II) follows from (b) and (c), recalling that each A has the form $(y.X)$ and, for all Y, $A\_X = X[Y/y]$. $\square$

Th. 3.5 is looser than a definition principle, since it does not state uniqueness of H. In effect, it is a "loose definition" principle, which makes no commitment to the choice of interpreting the variables. (Though it can be proved that H is uniquely determined by its action on variables. As a trivial example, the identity function on terms is uniquely identified by its

action on variables and by equations (I) and (II). Other functions, such as term-depth, do *not* fall into the cardinality hypothesis of this proposition, but of course can be defined using Prop. 3.4.) Note the "impredicative" nature of equation (II): it "defines" H on $\mathsf{Lam}\,A$ in terms of the "HOAS-components" of A, where a "HOAS component" is a result of applying A (as a function) to a term X and can of course be larger than A. Note also that the type of cLam is rather restricted, making cLam instantiable only to commutative infinitary operators such as the logical quantifiers. This proposition can be useful in situations where the existence of a compositional map is the only relevant aspect, allowing to take a shortcut from the first-order route of achieving compositionality through interpretation in environments – our proof of Strong Normalization from Section 3.5 takes advantage of this.

Conclusion: While the above preparations for HOAS on top of FOAS do require some work, this work is uniformly applicable to any (statically-scoped) syntax with bindings, hence automatable. Moreover, once this definitional effort is finished, one can forget about the definitions and work entirely in the comfortable HOAS setting (meaning: no more α-representatives, variable capture, etc.), as illustrated next.

3.4 HOAS representation of inference

This section deals with the HOAS representation of inductively defined relations, such as typing and reduction. Given an inductively defined relation on the first-order syntax employing the first-order operators, we *transliterate* it through our HOAS view, roughly as follows:
(I) abstractions constructed by terms with explicit dependencies become "plain" abstractions (used as functions);
(II) terms with implicit dependencies become abstractions applied to the parameter they depend on;
(III) substitution becomes functional application;
(IV) unbound arbitrary variables become arbitrary terms;
(V) scope extrusion is handled by universal quantification.
(We explain and illustrate these as we go through the examples, where the informal notions of implicit and explicit dependency will also be clarified.)

Our presentation focuses on a particular example, the typing and reduction of System F, but the reader can notice that the approach is rather general, covering a large class of reduction and type systems.

At this point, the reader should recall the definitions and notations pertaining to System F from Section 3.2.2. Notations, in a nutshell: lowercases x, y, z for dvars, uppercases X, Y, Z for dterms, uppercases A, B for dabstractions, calligraphic uppercases $\mathcal{A}, \mathcal{B}$ for 2-dabstractions;

for the type versions of the above, we prefix everything by "t". All the discussion from Section 3.3 duplicates for the two copies of the λ-calculus that make the syntax of System F. In particular, we have data-abstraction-lifted operators $\mathsf{App} : \mathbf{dabs} \to \mathbf{dabs} \to \mathbf{dabs}$, $\mathsf{Lam} : \mathbf{dabs2} \to \mathbf{dabs}$, etc. (where **dabs2** is the set of data 2-abstractions).

3.4.1 Representation of reduction

We define $\rightsquigarrow\!\!\!\!\rightarrow : \mathbf{dterm} \to \mathbf{dterm} \to \mathbf{bool}$ inductively, by the following clauses:

$$\frac{\cdot}{\mathsf{App}\ (\mathsf{Lam}\ A)\ X \rightsquigarrow\!\!\!\!\rightarrow A\_X}(\mathsf{HBeta}) \qquad \frac{\forall Z.\ A\_Z \rightsquigarrow\!\!\!\!\rightarrow B\_Z}{\mathsf{Lam}\ A \rightsquigarrow\!\!\!\!\rightarrow \mathsf{Lam}\ B}(\mathsf{HXi})$$

$$\frac{X \rightsquigarrow\!\!\!\!\rightarrow Y}{\mathsf{App}\ X\ Z \rightsquigarrow\!\!\!\!\rightarrow \mathsf{App}\ Y\ Z}(\mathsf{HAppL}) \qquad \frac{X \rightsquigarrow\!\!\!\!\rightarrow Y}{\mathsf{App}\ Z\ X \rightsquigarrow\!\!\!\!\rightarrow \mathsf{App}\ Z\ Y}(\mathsf{HAppR})$$

Adequacy of the reduction representation is contained in:

Prop 3.6 *The following are equivalent:*
- *(1)* $X \rightsquigarrow Y$.
- *(2)* $X \rightsquigarrow\!\!\!\!\rightarrow Y$.
- *(3)* $\forall \rho \in \mathbf{denv}.\ X[\rho] \rightsquigarrow\!\!\!\!\rightarrow Y[\rho]$.

Proof sketch.
(1) implies (3): By fresh induction, i.e., Prop. 2.5 from Section 2.2.2, taking the parameter to be ρ.
(3) implies (2): Immediately, taking ρ to be the identity environment.
(2) implies (1): By induction on the definition of $\rightsquigarrow\!\!\!\!\rightarrow$. $\square$

Remember that our HOAS representation dwells in the same universe as the original system, i.e., both the original relation $\rightsquigarrow$ and the representation relation $\rightsquigarrow\!\!\!\!\rightarrow$ act on the same syntax – they only differ *intensionally* in the way their definition manipulates this syntax: the former through bindings and substitution, the latter through abstractions-as-functions and function application. Looking for the incarnations of the general HOAS-transliteration patterns (I)-(V) listed at the beginning of this section, we find that:
- The definition of $\rightsquigarrow\!\!\!\!\rightarrow$ is obtained by modifying in $\rightsquigarrow$ *only the clauses involving binding and substitution*: (Beta), (Xi);
- In (Beta) and (Xi), $\mathsf{Lam}(x.Y)$, $\mathsf{Lam}(z.X)$ and $\mathsf{Lam}(z.Y)$ become $\mathsf{Lam}\ A$, $\mathsf{Lam}\ A$ and $\mathsf{Lam}\ B$, according to (I);
- In (Beta), $Y[X/x]$ becomes A_X, according to (III);
- In (Xi), *regarded as applied backwards*, we have the extrusion of the scope of z, as z is bound in the conclusion and free in the hypothesis – by pattern (V), this brings universal quantification over an arbitrary term Z in the hypothesis, as well as the acknowledgement of

an implicit dependency on z (now having become Z) in the X and Y from the hypothesis, making them become, by (II), abstractions applied to the implicit parameter, A_Z and B_Z.
(Note that this example does not illustrate pattern (IV), since all variables appearing in the definition of $\rightsquigarrow$ are bound.)[4]

The infinitary clause (HXi) from the definition of $\rightsquigarrow\!\!\!\!\rightarrow$ (whose premise quantifies over all dterms Z) is convenient when proving that $\rightsquigarrow\!\!\!\!\rightarrow$ *is included in* another relation, as it makes a very strong induction hypothesis, much stronger than that given by (Xi) for $\rightsquigarrow$. This is also true for rule inversion, where from $\mathsf{Lam}\ A \rightsquigarrow\!\!\!\!\rightarrow \mathsf{Lam}\ B$ we can infer a good deal of information compared to the first-order case. However, when proving that $\rightsquigarrow\!\!\!\!\rightarrow$ *includes* a certain relation, it appears that a HOAS clause matching (Xi) more closely may help. Such a clause can be extracted from (Xi):

Prop 3.7 $\rightsquigarrow\!\!\!\!\rightarrow$ *is closed under the following rule:*

$$\frac{\mathsf{fresh}\ z\ A \quad \mathsf{fresh}\ z\ B \quad A\_z \rightsquigarrow\!\!\!\!\rightarrow B\_z}{\mathsf{Lam}\ A \rightsquigarrow\!\!\!\!\rightarrow \mathsf{Lam}\ B}(\mathsf{HXi'})$$

Proof sketch. Assume that z is fresh for A and B and that $A\_z \rightsquigarrow\!\!\!\!\rightarrow B\_z$. By the freshness of z, we obtain X and Y such that $A = (z.X)$ and $B = (z.Y)$. Then $A\_z = X$ and $B\_z = Y$, and therefore the desired fact, $\mathsf{Lam}\ A \rightsquigarrow\!\!\!\!\rightarrow \mathsf{Lam}\ B$, follows from adequacy together with the closedness of $\rightsquigarrow$ under (Xi). □

(Since $\rightsquigarrow$ turns out to preserve freshness, the above proposition can actually be strengthened by renouncing the $\mathsf{fresh}\ z\ B$ hypothesis; but in this presentation we focus on facts that follow from the HOAS style of representation and adequacy, and not from specifics of the given relation.)

Note that (HXi') is stronger than (HXi) (but stronger as a rule means weaker as an induction-principle clause). A rule such as (HXi') should be viewed as a facility to descend, if necessary, from the HOAS altitude into "some details" (here, a freshness side-condition). This fits into our goal of encouraging HOAS definitions and proofs, while also allowing access to details on a by-need basis.

Since, by Prop. 3.6, the relations $\rightsquigarrow$ and $\rightsquigarrow\!\!\!\!\rightarrow$ coincide, hereafter we shall use only the symbol "$\rightsquigarrow\!\!\!\!\rightarrow$".

3.4.2 Representation of inference

A *HOAS context* (*Hcontext* for short) $\Delta \in \mathbf{Hctxt}$ is a list of pairs in $\mathbf{dterm} \times \mathbf{tterm}$, $X_1 : tX_1, \ldots, X_n : tX_n$. Note that $\mathbf{ctxt} \subseteq \mathbf{Hctxt}$.

[4] What we discuss here, in the context of the aforementioned patterns, are not the inductively defined relations, but the inductive definitions themselves; and what we loosely refer to as "variables" and "terms" appearing in these definitions are really variable and term meta-variables.

For Hcontexts, freshness, $\mathsf{fresh} : \mathbf{dvar} \to \mathbf{Hctxt} \to \mathbf{bool}$ and $\mathsf{fresh} : \mathbf{tvar} \to \mathbf{Hctxt} \to \mathbf{bool}$, and parallel substitution, $\_[\_,\_] : \mathbf{Hctxt} \to \mathbf{tenv} \to \mathbf{denv} \to \mathbf{Hctxt}$ are defined recursively as expected:

- $\mathsf{fresh}\, y\, [] = \mathsf{True}$;
- $\mathsf{fresh}\, y\, (\Delta, (X : tX)) = (\mathsf{fresh}\ y\ \Delta \wedge \mathsf{fresh}\ y\ X)$;
- $\mathsf{fresh}\ ty\ [] = \mathsf{True}$;
- $\mathsf{fresh}\ ty\ (\Delta, (x : tX)) = (\mathsf{fresh}\ ty\ \Delta \wedge \mathsf{fresh}\ ty\ tX)$;
- $[]\,[\xi, \rho] = []$;
- $(\Delta, (X : tX))\,[\xi, \rho] = (\Delta[\xi, \rho],\ (X[\rho] : tX[\xi]))$.

We represent type inference by the relation $(\_ \vdash\!\!\!\vdash \_ : \_) : \mathbf{Hctxt} \to \mathbf{dterm} \to \mathbf{tterm} \to \mathbf{bool}$, called *HOAS typing* (*Htyping* for short), defined inductively by the following clauses:

$$\frac{\cdot}{\Delta, X : tX \vdash\!\!\!\vdash X : tX}(\mathsf{HAsm}) \qquad \frac{\Delta \vdash\!\!\!\vdash X : tX}{\Delta, Y : tY \vdash\!\!\!\vdash X : tX}(\mathsf{HWeak})$$

$$\frac{\forall X.\ \Delta, X : tX \vdash\!\!\!\vdash A\ \_\ X : tY}{\Delta \vdash\!\!\!\vdash \mathsf{Lam}\ A : \mathsf{Arr}\ tX\ tY}(\mathsf{HArrI}) \qquad \frac{\forall tX.\ \Delta \vdash\!\!\!\vdash Y : tA\ \_\ tX}{\Delta \vdash\!\!\!\vdash Y : \mathsf{Al}\ tA}(\mathsf{HAlI})$$

$$\frac{\Delta \vdash\!\!\!\vdash X : \mathsf{Arr}\ tY\ tZ \qquad \Delta \vdash\!\!\!\vdash Y : tY}{\Delta \vdash\!\!\!\vdash \mathsf{App}\ X\ Y : tZ}(\mathsf{HArrE}) \qquad \frac{\Delta \vdash\!\!\!\vdash Y : \mathsf{Al}\ tA}{\Delta \vdash\!\!\!\vdash Y : tA\ \_\ tX}(\mathsf{HAlE})$$

Prop 3.8 *(Adequacy) The following are equivalent:*

- *(1)* $\Gamma \vdash X : A$.
- *(2)* $\Gamma \vdash\!\!\!\vdash X : A$. *(Note: contexts are particular Hcontexts.)*
- *(3)* $\Gamma[\xi, \rho] \vdash\!\!\!\vdash X[\rho] : A[\xi]$ *for all* $\xi \in \mathbf{tenv}$ *and* $\rho \in \mathbf{denv}$.

Proof sketch. Similarly to Prop. 3.6 □

It follows from Prop. 3.8 that $\vdash\!\!\!\vdash$ is a *conservative extension* (from contexts to Hcontexts) of $\vdash$. Thus, unlike with reduction, our HOAS representation of typing, $\vdash\!\!\!\vdash$, does *not* manipulate the same items as the original relation $\vdash$, but *extends* the domain – essentially, the new domain is the closure of the original domain under parallel substitution. Hereafter we write $\vdash\!\!\!\vdash$ for either relation, but still have Γ range over **ctxt** and Δ over **Hctxt**.

The only pattern from (I)-(V) exhibited by our HOAS- transliteration of typing that is not already present in the one for reduction is (IV), shown in the transliterations of (Asm), (Weak) and (ArrI) – there, we have the variables x and y becoming terms X and Y in (HAsm) (HWeak) and (HArrI). At (ArrI), (IV) is used in combination with (V), because x is also extruded back from the conclusion to the hypothesis, thus becoming in the hypothesis of (HArrI) a universally quantified term X. Another phenomenon not exhibited by reduction is the presence of freshness side-conditions (in the original system), whose effect is to *prevent dependencies* – e.g., the side-condition $\mathsf{fresh}\ y\ \Gamma$ from (Weak) says that Γ does not depend on x, meaning that, when transliterating (Weak) into (HWeak), (II) is not applicable to Γ. (Otherwise, to represent this we would need Hcontext-abstractions!)

Note that $\rightsquigarrow\!\!\!\!\rightarrow$ and $\rightsquigarrow$ coincide, while $\vdash\!\!\!\!\vdash$ is only a conservative extension of $\vdash$ – this is because our HOAS transliteration method *always closes under parallel substitution*, and $\rightsquigarrow$, unlike $\vdash$, is already closed. The presence of unbound variables in the first-order definition, requiring modification (IV), is a precise indicator of non-closedness.

3.4.3 Induction principle for type inference

By definition, $\vdash\!\!\!\!\vdash$ offers an induction principle: If a relation $R : \mathbf{Hctxt} \to \mathbf{dterm} \to \mathbf{tterm} \to \mathbf{bool}$ is closed under the rules defining $\vdash\!\!\!\!\vdash$, then $\forall \Delta, X, tX.\ \Delta \vdash\!\!\!\!\vdash X : tX \Longrightarrow R\ \Delta\ X\ tX$.

A HOAS technique should ideally do away (whenever possible) not only with the explicit reference to bound variables and substitution, but with the explicit reference to inference (judgment) contexts as well. Our inductive definition of Htyping achieves the former, but not the latter. Now, trying to naively eliminate contexts in a "truly HOAS" fashion, replacing, e.g., the rule (HArrI) with something like:

$$\frac{\forall X.\ \mathsf{typeOf}\ X\ tX \Longrightarrow \mathsf{typeOf}\ (A\ \_\ X)\ tY}{\mathsf{typeOf}\ (\mathsf{Lam}\ A)\ (\mathsf{Arr}\ tX\ tY)} \quad (*)$$

in an attempt to define *non-hypothetic typing* (i.e., typing in the empty context) directly as a binary relation typeOf between dterms and tterms, we hit two well-known problems:
-(I) The contravariant position of $\mathsf{typeOf}(X, tX)$ prevents the clause (*) from participating at a valid inductive definition.
-(II) Even if we "compromise" for a non-definitional (i.e., axiomatic) approach, but would like to retain the advantages of working in a standard logic, then (*) is likely to *not be sound*, i.e., not capture correctly the behavior of the original system. Indeed, in a classical logic it would allow one to type any $\mathsf{Lam}\,A$ to a type $\mathsf{Arr}\ tX\ tY$ for some non-inhabited type tX. Moreover, even we restrict ourselves to an intuitionistic setting, we still need to be very careful with (and, to some extent, make compromises on) the foundations of the logic in order for axioms like (*) to be sound. This is because, while the behavior of the intuitionistic connectives accommodates such axioms adequately, other mechanisms pertaining to recursive definitions are not a priori guaranteed to preserve adequacy – see [67, 81].

So what can one make of a clause such as (*) in a framework with meta-reasoning capabilities? As already discussed in the introduction, the HOAS-tailored framework's solution is axiomatic: (*) would be an axiom in a logic L (hosting the representation of the object system), with L itself is viewed as an object by the meta-logic; in the meta-logic then, one can perform proofs by induction on derivations in L. Thus, HOAS-tailored frameworks solve the problems with (*) by stepping one level up to a meta-logic. Previous work in general-purpose frameworks, after several experiments, eventually proposed similar solutions, either of directly interfering with the framework axiomatically [94] or of employing the

mentioned intermediate logic L [92].

Our own solution has an entirely different flavor, and does not involve traveling between logics and/or postulating axioms, but stays in this world (the same mathematical universe where all the development has taken place) and sees what this world has to offer: it turns out that clauses such as (*) are "backwards sound", in the sense that any relation satisfying them will include the empty-context Htyping relation. This yields "context-free" induction:

Theorem 3.9 *Assume* $\theta : \mathbf{dterm} \to \mathbf{tterm} \to \mathbf{bool}$ *such that the following hold:*

$$\frac{\forall X.\ \theta\ X\ tX \implies \theta\ (A\ \_\ X)\ tY}{\theta\ (\mathsf{Lam}\ A)\ (\mathsf{Arr}\ tX\ tY)}(\mathsf{Arrl}_\theta) \qquad \frac{\forall tX.\ \theta\ Y\ (tA\ \_\ tX)}{\theta\ Y\ (\mathsf{Al}\ tA)}(\mathsf{All}_\theta)$$

$$\frac{\theta\ Y\ (\mathsf{Arr}\ tX\ tZ) \qquad \theta\ X\ tX}{\theta\ (\mathsf{App}\ Y\ X)\ tZ}(\mathsf{ArrE}_\theta) \qquad \frac{\theta\ Y\ (\mathsf{Al}\ tA)}{\theta\ Y\ (tA\ \_\ tX)}(\mathsf{AlE}_\theta)$$

Then $\vdash X : tX$ *implies* $\theta\ X\ tX$ *for all* X, tX*. (In other words,* θ *includes the non-hypothetic Htyping relation.)*

Proof sketch. We take $R : \mathbf{Hctxt} \to \mathbf{dterm} \to \mathbf{tterm} \to \mathbf{bool}$ to be
- $R\ \Delta\ X\ tX = ((\forall\ (Y : tY) \in \Delta.\ \theta\ Y\ tY) \implies \theta\ X\ tX)$.
Then R satisfies the clauses that define $\vdash$, hence, in particular, for all X, tX, $\vdash X : tX$ implies $R\ []\ X\ tX$, i.e., $\theta\ X\ tX$. $\square$

Interestingly, a simple adaptation of this "context-free" technique works for proving contextual properties too, if we remember that the contexts we really care about in the end are the original contexts **ctxt**, not the HOAS contexts **Hctxt**: (Indeed, having formulated adequacy in the same meta-logic where we specified the object system helps us keep in mind that HOAS contexts are a more general, but merely auxiliary notion.)

Theorem 3.10 *Assume* $\theta : \mathbf{dterm} \to \mathbf{tterm} \to \mathbf{bool}$ *satisfies the conditions from Th. 3.9, and, in addition, has the property that* $\theta\ x\ tX$ *holds for all* $x \in \mathbf{dvar}$ *and* $tX \in \mathbf{tterm}$*.*

Then $\Gamma \vdash X : tX$ *implies* $\theta\ X\ tX$ *for all* Γ, X, tX*.*

Proof sketch. We take R as in the proof of Th. 3.9. Then again R satisfies the clauses that define $\vdash$, hence it includes $\vdash$, and in particular the following holds for all Γ, X, tX:
- $\Gamma \vdash X : tX \wedge (\forall\ (y : tY) \in \Gamma.\ \theta\ y\ tY) \implies \theta\ X\ tX$,
which means, using the extra hypothesis, that
- $\Gamma \vdash X : tX \implies \theta\ X\ tX$,
as desired $\square$

Viewing relations as nondeterministic functions, we can rephrase the last two theorems in a manner closer to the intuition of types as sets of data, with a *logical predicate* [143] flavor:

Theorem 3.9 (rephrased) *Assume* $\theta : \mathbf{dterm} \to \mathbf{P}(\mathbf{tterm})$ *such that:*

$$\frac{\forall X.\ X \in \theta\ tX \implies (A\ \_\ X) \in \theta\ tY}{(\mathsf{Lam}\ A) \in \theta\ (\mathsf{Arr}\ tX\ tY)}(\mathsf{Arrl}_\theta) \qquad \frac{\forall tX.\ Y \in \theta\ (tA\ \_\ tX)}{Y \in \theta\ (\mathsf{Al}\ tA)}(\mathsf{All}_\theta)$$

$$\frac{Y \in \theta\ (\mathsf{Arr}\ tX\ tZ) \qquad X \in \theta\ tX}{(\mathsf{App}\ Y\ X) \in \theta\ tZ}(\mathsf{ArrE}_\theta) \qquad \frac{Y \in \theta\ (\mathsf{Al}\ tA)}{Y \in \theta\ (tA\ \_\ tX)}(\mathsf{AlE}_\theta)$$

Then $\vdash X : tX$ *implies* $X \in \theta\ tX$ *for all* X, tX.

Theorem 3.10 (rephrased) *Assume* $\theta : \mathbf{dterm} \to \mathbf{tterm} \to \mathbf{bool}$ *satisfies the conditions from Th. 3.9 (rephrased), and, in addition, has the property that* $x \in \theta\ tX$ *holds for all* $x \in \mathbf{dvar}$ *and* $tX \in \mathbf{tterm}$.

Then $\Gamma \vdash X : tX$ *implies* $X \in \theta\ tX$ *for all* Γ, X, tX.

3.5 The HOAS principles at work

In this section we give a proof of strong normalization for System F within our HOAS representation using the developed definitional and proof machinery.

Remember that when introducing System F in Section 3.2.2 we fixed *infinite* sets of type and data variables, **dvar** and **tvar**, without making other assumption about their cardinalities. But now we commit to such an assumption, asking that we have much more type variables than data variables, namely, that **tvar** has a cardinality greater than or equal to that of $\mathbf{P}(\mathbf{dvar})$. (This assumption is needed for obtaining a compositional map via Th. 3.5.) One can easily see that this assumption does not affect the generality of the result, since once strong normalization has been proved for *some* fixed infinite cardinalities of the variable sets, then it can be inferred that it holds for *any* other infinite cardinalities – moreover, this also seems to be the case for most of the interesting properties considered for typing systems in the literature. Note also that this cardinality assumption has an intuitive reading in Cantorian set theory: think of types as sets of data, identify types with tterms and data with dterms; then, saying that $\mathsf{card}(\mathbf{tvar}) = \mathsf{card}(\mathbf{P}(\mathbf{dvar}))$ is the same as saying that $\mathsf{card}(\mathbf{tterm}) = \mathsf{card}(\mathbf{P}(\mathbf{dterm}))$, i.e., that types are indeed (in bijection with) sets of data.

3.5.1 An effective proof principle for typable terms

Before going after a proof of a particular property of System F, we first analyze how we could hypothetically employ our HOAS machinery in a potential proof.

A typical property φ that needs to be proved for typable terms is of course of the same (meta)type as the typing relation $\vdash$ itself, namely $\mathbf{ctxt} \to \mathbf{dterm} \to \mathbf{tterm} \to \mathbf{bool}$. However, many relevant properties $\varphi\ \Gamma\ X\ tX$ are oblivious to the context Γ, and some even to the tterm tX. For instance:
- Strong normalization: $\varphi : \mathbf{dterm} \to \mathbf{bool}$, $\varphi\ X =$ "X is strongly normalizing";
- Church-Rosser: $\varphi : \mathbf{dterm} \to \mathbf{bool}$, $\varphi\ X = (\forall Y_1, Y_2.\ X \twoheadrightsquigarrow Y_1 \wedge X \twoheadrightsquigarrow Y_1 \Longrightarrow (\exists Z.\ Y_1 \twoheadrightsquigarrow Z \wedge Y_2 \twoheadrightsquigarrow Z))$.

(Considering the latter property w.r.t. the typed terms is of course only interesting in cases where it does not hold for untyped terms already, e.g., Church-style type systems versus $\beta\eta$-reduction.) For such cases, our HOAS machinery comes very handy, as we show below.

So, given $\varphi : \mathbf{dterm} \to \mathbf{bool}$, how would one go about proving that $\Gamma \vdash X : tX$ implies $\varphi\ X$ for all Γ, X, tX? It will be more convenient to replace φ with its associated set $G = \{X.\ \varphi\ X\}$, and therefore the question becomes: Given $G \in \mathbf{P}(\mathbf{dterm})$, how would one go about proving that $\Gamma \vdash X : tX$ implies $X \in G$ for all Γ, X, tX? A "first reflex" is to use the HOAS-induction principle from Th. 3.10 (rephrased), that is, search for $\theta : \mathbf{tterm} \to \mathbf{P}(\mathbf{dterm})$ with $\forall tX.\ \theta\ tX \subseteq G$, $\forall tX, X.\ X \in \theta\ tX \Longrightarrow \varphi\ X$, i.e., $\theta : \mathbf{tterm} \to \mathbf{P}(G)$, satisfying the clauses from there. Then Th. 3.5 suggests a HOAS-recursive definition of θ. After some investigation, we are naturally led to a general criterion justifiable by the combination of HOAS induction and recursion (in what follows, we let Zs range over lists of terms and take $\mathsf{AppL} : \mathbf{dterm} \to \mathbf{List}(\mathbf{dterm}) \to \mathbf{dterm}$ to be defined by $\mathsf{AppL}\ X\ [] = X$ and $\mathsf{AppL}\ X\ (Z, Zs) = \mathsf{AppL}\ (\mathsf{App}\ X\ Z)\ Zs$; moreover, given a list Zs and a set G, we loosely write $Zs \subseteq G$ to indicate that all terms from Zs are in G):

Prop 3.11 *Assume that $G \subseteq \mathbf{dterm}$ such that the following hold:*

$$\frac{Zs \subseteq G}{\mathsf{AppL}\ y\ Zs \in G}(\mathsf{VCl}^G) \qquad \frac{\forall x.\ \mathsf{App}\ Y\ x \in G}{Y \in G}(\mathsf{AppCl}^G)$$

$$\frac{X \in G \qquad Zs \subseteq G \qquad \mathsf{AppL}\ (A\ \_\ X)\ Zs \in G}{\mathsf{AppL}\ (\mathsf{App}\ (\mathsf{Lam}\ A)\ X)\ Zs \in G}(\mathsf{Cl}^G)$$

Then $\Gamma \vdash X : tX$ implies $X \in G$ for all Γ, X, tX.

Proof sketch. Consider the following clauses, expressing potential properties of subsets $S \subseteq \mathbf{dterm}$ (assumed universally quantified over all the other parameters):
- (VCl^S): if $Zs \subseteq G$, then $\mathsf{AppL}\ y\ Zs \in S$;
- (Cl^S): if $X \in G$, $Zs \subseteq G$ and $\mathsf{AppL}\ (A\ \_\ X)\ Zs \in S$, then $\mathsf{AppL}\ (\mathsf{App}\ (\mathsf{Lam}\ A)\ X)\ Zs \in S$.

Let $\mathcal{C} = \{S \subseteq G.\ (\mathrm{VCl}^S) \text{ and } (\mathrm{Cl}^S) \text{ hold}\}$. We define $\mathsf{cArr} : \mathcal{C} \to \mathcal{C} \to \mathcal{C}$ and $\mathsf{cAl} : \mathbf{P}_{\neq\emptyset}(\mathcal{C}) \to \mathcal{C}$ by $\mathsf{cArr}\ S_1\ S_2 = \{Y.\ \forall X \in S_1.\, \mathsf{App}\ Y\ X \in S_2\}$ and $\mathsf{cAl}\ K = \bigcap K$.
(Checking that these operations are well-defined is routine – note that, while for cAl well-definedness follows simply by the fact that (VCl^S) and (Cl^S) are in the Horn format, the

actual content of these clauses is crucial for the well-definedness of cArr.)

By Th. 3.5, there exists a map $\theta : \mathbf{tterm} \to \mathcal{C}$ that commutes with cArr and cAl, i.e.:
-(I) $\theta(\mathsf{Arr}\ tX\ tZ) = \{Y.\ \forall X \in \theta\ tX.\ \mathsf{App}\ Y\ X \in \theta\ tZ\}$.
-(II) $\theta(\mathsf{Al}\ tA) = \bigcap_{tX \in \mathbf{tterm}} \theta(tA\ \_\ tX)$.

Now, (II) is precisely the conjunction of the clauses (AlI_θ) and (AlE_θ) from Th. 3.9 (rephrased), while the left-to-right inclusion part of (I) is a rephrasing of (ArrE_θ). Finally, (AlE_θ) holds because (Cl^S) holds for all $S \in \mathcal{C}$. Thus, the hypotheses of Th. 3.9 (rephrased) are satisfied by $\theta : \mathbf{tterm} \to \mathcal{C}$ (regarded as a map in $\mathbf{tterm} \to \mathbf{P}(\mathbf{dterm})$). Hence, $\forall X, tX.\ \vdash X : tX \Longrightarrow X \in \theta\, tX$. And since $\forall tX.\, \theta\ tX \subseteq G$, we get $\forall X, tX.\ \vdash X : tX \Longrightarrow X \in G$. $\square$

We call a subset $G \subseteq \mathbf{dterm}$ *type-closed* (terminology taken from [89]) if it satisfies the hypotheses of Prop. 3.11.

3.5.2 Proof of strong normalization for System F

We let $\mathcal{SN}$ be the set of all strongly normalizing dterms.

Prop 3.12 *(Strong Normalization) If* $\Gamma \vdash X : tX$, *then* $X \in \mathcal{SN}$.

Proof. One can verify that $\mathcal{SN}$ is type-closed – the verification goes smoothly (provided one has proved the substitution lemma indicated below), but is tedious. Then, the result follows from Prop. 3.11. $\square$

The latter proposition employs the following lemma, whose proof occasions the usage of the argument-permutative induction from Prop. 3.3:

Prop 3.13 *If* $X \rightsquigarrow^* X'$, *then* $A\ \_\ X \rightsquigarrow^* A\ \_\ X'$.

Proof. First, we note that $\mathsf{fresh}\ z\ A \wedge A\ \_\ z \rightsquigarrow^* A'\ \_\ z \Longrightarrow \mathsf{Lam}\ A \rightsquigarrow^* \mathsf{Lam}\ A'$, from which we get $(\forall z.\ A\ \_\ z \rightsquigarrow^* A'\ \_\ z) \Longrightarrow \mathsf{Lam}\ A \rightsquigarrow^* \mathsf{Lam}\ A'$ (**)

Now, we employ the principle from Prop. 3.3, performing induction on A. For the only interesting case, assume A has the form $\mathsf{Lam}\ \mathcal{A}$. We know from IH that $\forall z.\ (\mathcal{A}\ \_1\ z)\_X \rightsquigarrow^* (\mathcal{A}\ \_1\ z)\_X' \wedge (\mathcal{A}\ \_2\ z)\_X \rightsquigarrow^* (\mathcal{A}\ \_2\ z)\_X'$. The second conjunct gives $\forall z.(\mathcal{A}\ \_1\ X)\_z \rightsquigarrow^* (\mathcal{A}\ \_1\ X')\_z$, hence, with (**), $\mathsf{Lam}(\mathcal{A}\ \_1\ X) \rightsquigarrow^* \mathsf{Lam}(\mathcal{A}\ \_1\ X')$, i.e., $(\mathsf{Lam}\,\mathcal{A})\ \_\ X \rightsquigarrow^* (\mathsf{Lam}\,\mathcal{A})\ \_\ X'$. (We also used the exchange and commutation laws from Section 3.3.2.) $\square$

The above proof reveals an interesting phenomenon: in a HOAS setting, where bindings are kept implicit and substitution is mere function application, in some proofs one may

need to perform a permutation of the "placeholders" for function application (requiring 2-abstractions), whereas in a first-order framework one would be able to proceed more directly.

Indeed, consider a first-order version of Prop. 3.13, stating that $\rightsquigarrow^*$ is substitutive: $X \rightsquigarrow^* X'$ implies $Y[X/x] \rightsquigarrow^* Y[X'/x]$. Its proof goes by fresh induction (Prop. 2.5) on Y with x, X, X' as parameters (thus taking **param** to be **var** $\times$ **term**), treating the case of abstraction as follows: Assume $Y = \mathsf{Lam}(z, Z)$ and z is fresh for x, X, X'. By IH, $Z[X/x] \rightsquigarrow^* Z[X'/x]$. By (Xi) (iterated), $\mathsf{Lam}(z.(Z[X/x])) \rightsquigarrow^* \mathsf{Lam}(z.(Z[X'/x]))$, hence (since z is fresh), $\mathsf{Lam}(z.Z)[X/x] \rightsquigarrow^* \mathsf{Lam}(z.Z)[X'/x]$, as desired.

Here, the proof of the first-order version of the fact is more direct than that of the HOAS version because under the first-order view a term Y allows substitution *at any position*, i.e., at any of its free variables, while under the HOAS view an abstraction A has only *one particular position* "prepared" for substitution. Therefore, whenever "multi-substitutive" reasoning is necessary, e.g., in the appearance of Z above with both x and z to be substituted during the analysis of the inductive case, under the HOAS view we need 2-*abstractions*, like $\mathcal{A}$ in the proof of Prop. 3.13, and employ the permutative "_2" application. Our definitional framework accommodates both the first-order and the HOAS proofs, since *the object syntax is the same*, being only subjected to two distinct views.

3.6 Formalization

While we have insisted on the general pattern followed by our HOAS constructions and results, they have not been developed mathematically or formally in a general context such as the one for our FOAS theory (arbitrary syntax with bindings, etc.). Currently, we have only formalized precisely the results stated, for their particular syntax.

The diagram in Figure 3.1 shows the relevant part of our theory structure in Isabelle. In fact, the part consisting of the theories D and T and the ones below them matches quite faithfully the sectionwise structure of this chapter and is conceptually self-contained.

Theories D and T instantiate, as usual, our general FOAS theory from Chapter 2 to the two copies of λ-calculus that make the syntax of System F. Thus, these two theories correspond to the first part of the Section 3.2.

The theories HOAS_View_D and HOAS_View_T formalize the HOAS view of the syntax of dterms and tterms, respectively, thus matching Section 3.3. Next we refer to HOAS_View_D only (since HOAS_View_T is similar). The definitions of abstraction application and the other operators described informally in this text and claimed to be independent of representatives are first given in Isabelle by picking *some* representatives. E.g. the operators varOfAbs and termOfAbs pick *together* a representative (x, Y) for an abstraction A, and then $A \_ X$ is defined to be $Y[X/x]$; then, "the real definition", not committing to any particular such pair, is

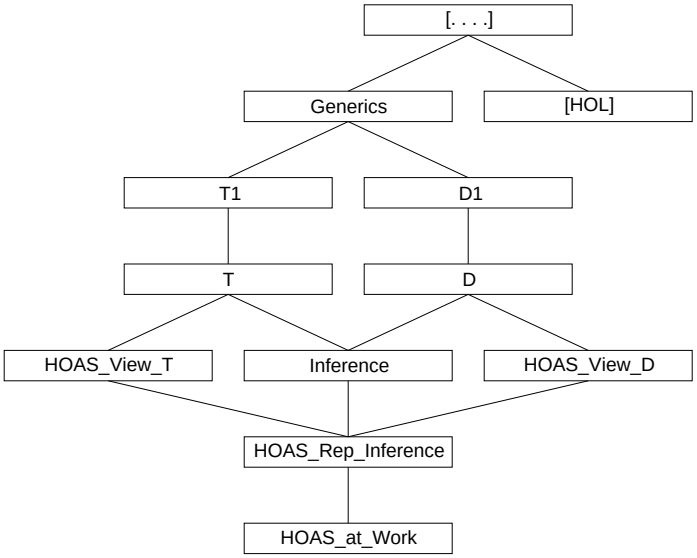

Figure 3.1: The relevant part of the theory structure in Isabelle

stated as a lemma: "$A\_X = Y[X/x]$ for *all* x, Y such that $A = (x.Y)$". (Note that in the scripts we write Dabs x Y for $(x.Y)$.) While the induction principles from Section 3.3.3 are rendered verbatim in the scripts, the formalizations of the recursive definition principles from Section 3.3.4 have a slightly different form, reflecting Isabelle's distinction between a type and a set. E.g., to obtain a flexible Isabelle version of Th. 3.5, we have the domain C from there represented not merely by a type, but by a type c together with a "well-structured-ness" predicate cWls : $c \to$ **bool**. Then a compositional map H as in Th. 3.5 is called there a "HOAS-morphism"; the existence of such a map is stated in the scripts as Th. ex_HOASmorph, and then rephrased as Th. ex_comp_, which matches Th. 3.5 more closely.

The theory HOAS_Rep_Inference formalizes the HOAS representation of inference, discussed in Section 3.4. The three subsections of this theory match those of Section 3.4. Our HOAS inference employs infinitary inductive clauses, but these are unproblematic in Isabelle, both definition-wise and proof-wise (of course, it is their uniformity that makes them unproblematic proof-wise). While in this chapter we assume **ctxt** $\subseteq$ **Hctxt**, in the formalization we have an explicit injection asHctxt : **ctxt** $\to$ **Hctxt**, and a predicate isCtxt : **Hctxt** $\to$ **bool** representing its image. As a "psychological" note, Isabelle is able to figure out automatically the proof of facts like Ths. 3.9 and 3.10 once we indicate the relation R, while for the human mind this is somewhat difficult to grasp, as is any statement whose justification involves implications nested on the left, as in $(\varphi \Rightarrow \chi) \Rightarrow \psi$, and finally we prove the result.

The theory HOAS_at_Work formalizes the strong normalization proof, corresponding to Section 3.5. Here is the content of this theory. First we prove the type-closedness criterion, Prop. 3.11. Then we prove Prop. 3.13 – we actually give two alternative proofs of this (FOAS and HOAS), reflecting our discussion following that proposition. Then we make further preparations for the proof of Prop. 3.12 in terms of some variations of the notion of reduction-simulation. Finally, we prove the result.

Our Isabelle scripts can be downloaded from [119]. The document SysF.pdf from that (zipped) folder contains a detailed presentation of the relevant theories. These theories can also be browsed in html format in the folder SysF_Browse (note that the browsable format shows also all the background (FOAS) theories needed for our HOAS work).

Here is a list of further differences between the text and the Isabelle scripts:

- freshAbs and freshEnv (instead of fresh) are used for the freshness operators on abstractions and environments, respectively.
- Dabs x X and Dabs2 x A (instead of $x.X$ and $x.A$) are used for the first-order dabstraction and 2-dabstraction constructs.
- Similarly, Tabs x tX and Tabs2 x tA (instead of $x.tX$ and $x.tA$) are used for the first-order tabstraction and 2-tabstraction constructs.
- In theories T and T1, since there is no overlap (yet) with data items, we do not prefix the variable names by "t".

- In the Isabelle scripts we have three kinds of notations for substitutions: arbitrary substitution in environments, $X[\rho]$, unary substitution ("usubst") $X[Y/y]$, and variable-for-variable unary substitution ("vusubst") $X[x//y]$; we also have (variable-for-variable) swapping, written $X[x \wedge y]$.
- While in this chapter we keep some injections implicit, in Isabelle we represent them explicitly:
— dVar : **dvar** → **dterm**, the injection of dvars as dterms;
— tVar : **tvar** → **tterm**, the injection of tvars as tterms;
— asHctxt : **ctxt** → **Hctxt**, the injection of contexts as Hcontexts;
— isCtxt : **Hctxt** → **bool**, the predicate checking if an Hcontext is (the image of) a **ctxt**.

3.7 Conclusions and related work

A goal of this chapter was to insist on, and bring technical evidence for, the advantage of using a general-purpose framework for HOAS, or, in other words, to employ HOAS within standard mathematics. We showed that our general-purpose framework offers access to some of the HOAS advanced conveniences, such as impredicative and context-free representations of (originally context-based) type systems.

Another goal was to bring, via an extensive HOAS exercise, more evidence to a belief seemingly shared by the whole HOAS community (beyond the large variety of proposed technical solutions), but not yet sustained by many examples in the literature (apart from those from [15]): that a HOAS representation of a system is in principle able not only to allow hassle-free manipulation and study of a system, but also to actually *shed more light on the deep properties of a system*. We believe that our general-purpose HOAS machinery does simplify and clarify the setting and justification of a notoriously hard result in type theory.

3.7.1 Work on HOAS

HOAS-tailored approaches. The HOAS-tailored framework approach yielded several theorem provers and functional programming environments (some of them already mature and with an extensive case-study record), including several extensions of LF – Twelf [6], Delphin [2], ATS [29], Beluga [113] – and Abella [4], a HOAS-specialized prover based on definitional reflection. is based on an extension of LF, where one can express certain $\forall\exists$ statements, the so-called *totality assertions* – here, the $\forall\exists$ extension is the meta-logical framework and LF is the logical framework. Beluga [113] is based on similar principles as Twelf. In Abella, the meta-logical framework is a (quasi) higher-order logic with definitional reflection and induction, and the meta-logic varies according to the need (variants of intuitionistic or linear higher-order or second-order logic being considered).

Hybrid. The Hybrid package [13], coming in an Isabelle/HOL and a Coq variant, is a successful realization of the general-purpose framework approach. Later versions of this system [92, 95, 43, 44] also import the three-level architecture idea from the HOAS-tailored framework approach.

Hybrid uses a representation of the λ-calculus very similar to ours. In the description below, we use the recent paper [43]. The collection of λ-terms is given there by the type **expr**, together with a predicate $\mathsf{proper} : \mathbf{expr} \to \mathbf{expr}$. (I.e., terms form the set $\{X \in \mathbf{expr}.\ \mathsf{proper}\ X\}$.)

The usual non-binding first-order operators are provided:
- $\mathsf{VAR} : \mathbf{var} \to \mathbf{expr}$, for variable-injection,
- $\mathsf{APP} : \mathbf{expr} \to \mathbf{expr} \to \mathbf{expr}$, for application,
- $\mathsf{CON} : \mathbf{const} \to \mathbf{expr}$, for constant injection (where **const** is an unspecified type of constants, as, e.g., in our Section 2.3.2).

All these operators are well-defined on λ-terms, i.e., preserve properness. Now, abstractions are defined as particular functions from expressions to expressions (predicate $\mathsf{abstr} : \mathbf{expr} \to \mathbf{expr}$), and a strong-HOAS binding operator $\mathsf{lambda} : (\mathbf{expr} \to \mathbf{expr}) \to \mathbf{expr}$ is introduced. Their set of abstractions, $\{f : \mathbf{expr} \to \mathbf{expr}.\ \mathsf{abstr}\ f\}$ can be seen to be isomorphic to our set **abs** of abstractions, by the bijection mapping any $A \in \mathbf{abs}$ to $\lambda X.\ A\_X$. Moreover, this bijection commutes with our Lam versus lambda. All these mean that our HOAS view is essentially the same as the Hybrid HOAS representation, the only difference being that we use first-order abstractions as functions, while they use an actual (restricted) space of functions.

After this point our approach and the Hybrid approach go different ways. For us, λ-calculus is an example of an object system, for which HOAS view and its various developed definition and proof principles are samples of the way in which any other object system would be treated. In Hybrid, the above λ-calculus encoding is not the typical encoding of an object-system, but part of a logical framework in its own right, aimed at representing object systems (see below).

Consequently, we pay λ-calculus more attention as an object system, i.e., w.r.t. reasoning mechanisms on its syntax. While
- our Prop. 3.2 is the same as the Hybrid MC-Theorem 6 from page 11 in op. cit.
- and our Prop. 3.4 being essentially the Gordon-Melham induction principle [58], is implicit in the Hybrid representation,

we go beyond these and provide:
- (1) support for inductive reasoning about abstractions – Prop. 3.3;
- (2) a strong-HOAS primitive recursor – Th. 3.5.[5]

[5] But see also the discussion of a primitive recursion principle introduced in [27].

(For instance, (1) was inspired by the need to perform variable-permutative reasoning about the object system, as in the proof of Prop. 3.13.) Moreover, at this "raw" mathematical level (the same where the λ-calculus syntax was defined and HOAS-represented), we discuss methodologies for HOAS-representing inference systems, reasoning about them and proving the adequacy of these representations.

On the other hand, within Hybrid one defines a second logical framework (besides, and inside, Isabelle itself), consisting of the following:
- Syntax (if we ignore the "False" connective $\bot$, which does not seem needed in any HOAS encoding):
— an unspecified set **atm** of *atoms* – these are supposed to be eventually instantiated with (first-order) statements about the terms of λ-calculus and (second-order) statements about abstractions, all determined by the syntactic constructs of the particular object system (to be encoded);
— the set **oo**, of *goals*, which are essentially conjunctions of Horn clauses over the atoms (thinking of the Horn clauses as including the outermost universal quantification);
— the set of *clauses*, which are essentially sets of universally quantified implications between goals and atoms (and are implicitly represented as pairs atom-goal).
(Thus, the clauses are a fragment of the Hereditary Harrop formulas [84]; these formulas are also considered in [126, 119], where they are called Horn^2 formulas.)
- Inference: natural-deduction intuitionistic rules, starting with a set of clauses as a fixed theory and inferring goals in contexts consisting of atomic assumptions (atoms).
(A linear-logic variation of the SL is also considered in [43], tailored for cases where the structural rules of weakening and contraction are not present in the object system, hence undesired at the meta-level either.)

Note that both Hybrid levels involve HOAS representations:
- (1) the representation of the λ-calculus terms with constants and of the SL logic over these terms in Isabelle;
- (2) the encoding of a given object system in SL.
An encoding as at (2) proceeds by indicating:
- some λ-calculus constants corresponding to the syntax of the object system;
- a set of clauses, essentially a Prolog program, as the aforementioned fixed theory for the SL inference, corresponding to the inference relations (such as typing, reduction etc.) of the object system.

The presence of these two levels allows for LF-style of encodings and proofs (while keeping the advantage of staying completely definitional – no axioms involved), with the proviso that the extra level of indirection needs to be "tamed" and kept on the background as much as possible – this is achieved by some Isabelle tactics. (The recent paper [45] compares Hybrid with HOAS-tailored LF-based systems.)

Back to level (1) though (which is essentially where we have been working in this chapter), [43] actually contemplates staying at this level, but concludes (on page 21) that this would be inconvenient, a main reason being the difficulty of keeping contexts implicit (as in LF) and, at the same time, having the necessary relations defined inductively. In Section 3.4.3, we have argued that this is, to some extent, possible, in that the relation can be defined inductively while a context-free version of induction can be extracted from this definition. Our Ths. 3.9 and 3.10 from there are stated for a particular typing system, but their pattern is quite general. As another instance of this pattern, consider the formal-verification case study employing an older version of Hybrid reported in [93]. It is about formalizing an instance of the method from [69] for proving applicative (bi)simulations to be (pre)congruences.

Let **clterm** be the set of *closed terms*, i.e., those for which all variables are fresh; an environment $\rho : \mathbf{var} \to \mathbf{term}$ will be called *closed* if its image consists of closed terms. The base relation is $\preceq \ : \mathbf{clterm} \to \mathbf{clterm} \to \mathbf{bool}$, the applicative simulation for the lazy λ-calculus, whose particular definition is not relevant to the next discussion (see [9, 69, 93]). The open applicative simulation, $\preceq^o \ : \mathbf{term} \to \mathbf{term} \to \mathbf{bool}$ is defined as the (closed-)substitutive coclosure of $\preceq$:

- $(X \preceq^o Y) = (\forall \rho \text{ closed}.\ X[\rho] \preceq Y[\rho])$.

Then the following inductive relation is introduced, which we define, following [93], as a contextual relation $(\_ \vdash \_ \preceq^\bullet \_) \ : \mathbf{ctxt} \to \mathbf{term} \to \mathbf{term} \to \mathbf{bool}$, where the contexts $\Gamma \in \mathbf{ctxt}$ are here non-repetitive lists of variables. Since we are concerned with formally certified adequacy as well, we first give the contextual version of the relation in its original, FOAS format (as defined in [69]):

$$\frac{x \preceq^o X'}{\Gamma \vdash x \preceq^\bullet X'}(\mathsf{Var}) \qquad \frac{\Gamma \vdash X \preceq^\bullet X' \qquad \Gamma \vdash Y \preceq^\bullet Y' \qquad \mathsf{App}\ X'\ Y' \preceq^o Z''}{\Gamma \vdash \mathsf{App}\ X\ Y \preceq^\bullet Z''}(\mathsf{App})$$

$$\frac{\Gamma, x \vdash Y \preceq^\bullet Y' \qquad \mathsf{Lam}(x.Y') \preceq^o Z''}{\Gamma \vdash \mathsf{Lam}(x.Y) \preceq^\bullet Z''}\ (\mathsf{Lam})\ [\mathsf{fresh}\ x\ \Gamma]$$

(This relation is called the *precongruence candidate*. As explained in [69] on page 105, $\Gamma \vdash X \preceq^\bullet X'$ should be read intuitively as: X' can be obtained from X via one bottom-up pass of replacements of subterms by terms that are larger under $\preceq^o$.)

In [93], the Hybrid encoding of this relation is given as a HOAS relation based on the same notion of context as list of variables. However, one can go further w.r.t. HOAS and use what we called "HOAS contexts" $\Delta \in \mathbf{Hctxt}$, here, lists of terms, obtaining the following relation $(\_ \vdash \_ \preceq^\bullet) : \mathbf{Hctxt} \to \mathbf{term} \to \mathbf{term} \to \mathbf{bool}$:

$$\frac{X \preceq^o X'}{\Delta \vdash X \preceq^\bullet X'}(\mathsf{HVar}) \qquad \frac{\Delta \vdash X \preceq^\bullet X' \qquad \Delta \vdash Y \preceq^\bullet Y' \qquad \mathsf{App}\ X'\ Y' \preceq^o Z''}{\Delta \vdash \mathsf{App}\ X\ Y \preceq^\bullet Z''}(\mathsf{HApp})$$

$$\frac{\forall X.\ \Delta, X \vdash A\ \_X \preceq^\bullet A'\ \_X \qquad \mathsf{Lam}\ A \preceq^o Z''}{\Delta \vdash \mathsf{Lam}\ A \preceq^\bullet Z''}(\mathsf{HLam})$$

The obvious adaptation of our adequacy result stated in Prop. 3.8 holds here too (crucial being the fact that the parameter of this definition, $\preceq^o$, is itself closed under substitution), implying that ($\_ \vdash \_ \preceq^\bullet$) is a conservative extension to HOAS contexts of ($\_ \vdash \_ \preceq^\bullet$). The above still uses contexts, but a context-free version of our Th. 3.10 holds here with a similar proof:

Assume $\theta : \mathbf{term} \to \mathbf{term} \to \mathbf{bool}$ *such that the following hold:*

$$\frac{X \preceq^o X'}{\theta\ X\ X'}(\mathsf{Var}_\theta) \qquad \frac{\theta\ X\ X' \qquad \theta\ Y\ Y' \qquad \mathsf{App}\ X'\ Y' \preceq^o Z''}{\theta\ (\mathsf{App}\ X\ Y)\ Z''}(\mathsf{App}_\theta)$$

$$\frac{\forall X.\ \theta\ (A\ \_\ X)\ (A'\ \_\ X) \qquad \mathsf{Lam}\ A \preceq^o Z''}{\theta\ (\mathsf{Lam}\ A)\ Z''}(\mathsf{Lam}_\theta)$$

Then $\Gamma \vdash X \preceq^o X'$ *implies* $\theta\ X\ X'$ *for all* Γ, X, X'.

Again, this (partly) fulfills the HOAS-purity goal of context-freeness discussed in our Section 3.4.3 and also in [43] (on page 21) and [93] (on page 7). (Although one can see that, unlike for the case of typing from Section 3.4.3, for have contexts have been slightly overused, as they do not hold information with "negative occurrences". Therefore we could have used non-contextual variants of ($\_ \vdash \_ \preceq^\bullet$) and ($\_ \vdash \_ \preceq^\bullet$) and a version of Prop. 3.6 to the same effect.)

Weak versus strong HOAS. A standard classification of HOAS approaches is in *weak* versus *strong* HOAS. Both capture object-level bindings by meta-level *functional* bindings; "weak" refers to the considered functions mapping *variables* to terms, while "strong" refers to these functions mapping *terms* to terms. Weak HOAS approaches are taken in [38, 68, 131, 61], including in category-theoretic form (with a denotational-semantics flavor) in [46, 67, 14, 47].[6] Our work in this chapter, the above HOAS-tailored approaches, as well as [39], the work on Hybrid [13, 92, 95, 43], as well as parametric HOAS [30], parametricity-based HOAS [70],[7] and de-Bruijn-mixed-HOAS [66], fall within strong HOAS.

In weak HOAS, some of the convenience is lost, since substitution of terms for variables is not mere function application, as in strong HOAS. On the other hand, weak HOAS is easier to define directly inductively. However, as illustrated in this paper and in previous work [39, 13], in a general-purpose setting having strong HOAS (perhaps on top of weak HOAS as in [39], or directly on top of the first-order syntax as here) is only a matter of some *definitional* work. Because variables are particular terms, strong HOAS can accommodate weak induction and recursion principles, and in fact in most situations only such weak

[6] Among these, the work on π-calculus [131, 47], although claimed as participating to weak HOAS, is in fact also strong HOAS, since channel terms coincide there with channel variables/names.

[7] The import of the notion of parametricity into HOAS was apparently pioneered by [141, 40].

principles are available due to the need of well-foundedness – Props. 3.2 and 3.3 are examples of "weak" principles within strong HOAS.

Our Th. 3.5 is a "strong" iteration principle, because it computes the value on $\mathsf{Lam}\ A$ in terms of the values of $A\ \_\ X$ (for all terms X), and thus treats A as if is a function from $\mathbf{term} \to \mathbf{term}$, as in strong HOAS. A "strong" iterator is also introduced in [27] (within a preliminary Coq-based version of one-level Hybrid). There, one works with de Bruijn terms as in Hybrid (described above), but unlike in Hybrid terms are allowed to have dangling references (the predicate that cuts them down does not prevent this) – let us denote their set by $\mathbf{term}_d$. Working with $\mathbf{term}_d$ rather than $\mathbf{term}$ allows one to define:

- $\mathsf{tbody} : (\mathbf{term}_d \to \mathbf{term}_d) \to \mathbf{term}_d$, such that, for each function $f : \mathbf{term}_d \to \mathbf{term}_d$ representing an actual abstraction A, $\mathsf{tbody}\ f$ (the "body" of f) is essentially the body of A with the bound variable replaced by the index 0;
- $\mathsf{funt} : (\mathbf{term}_d \to \mathbf{term}_d) \to (\mathbf{term}_d \to \mathbf{term}_d)$, "normalizing" any function to one representing an abstraction – the only relevant property of funt is that, if $f : \mathbf{term}_d \to \mathbf{term}_d$ represents an abstraction A, then $\mathsf{funt}\ f = f$;
- $\mathsf{Fun} : (\mathbf{term}_d \to \mathbf{term}_d) \to \mathbf{term}_d$ – this is the higher-order binding operator, essentially lambda from the above discussion on Hybrid.

A recursor is then defined, whose associated iterator,[8] which we shall call Iter, has type (given the target domain B) $(\mathbf{var} \to B) \to (I\!N \to B) \to (B \to B \to B) \to (B \to B) \to (\mathbf{term}_d \to B)$ – this is the same as the type of the (locally nameless) first-order de Bruijn iterator. What makes it strong HOAS is the equation for the abstraction case (when defining a presumptive $F : \mathbf{term}_d \to B$), where the de-Bruijn-required element is produced using tbody:

- $F\ (\mathsf{Fun}\ f) = B_{\mathrm{lam}}\ (\mathsf{tbody}\ f)$.

Thus, a strong-HOAS function $f : \mathbf{term}_d \to \mathbf{term}_d$ appears on the left-hand side, although this higher order type is not reflected in the structure on the target domain. This approach has some similarities with the one from [14], discussed in Section 2.10 – like there, one combines an essentially first-order iterator with the advantage of a higher-order "notation"; the difference is that here one has strong HOAS, not weak HOAS. Compared to our own iterator from Th. 3.5, Iter does not have to deal with the problem of impredicativity, since a reduction to the standardly well-defined de Bruijn iteration is performed explicitly via tbody. The flip side is that no impredicative flavor is given back either, and therefore, e.g., the environment-free definition of our type interpretation θ from the proof of Prop. 3.11 is not covered by Iter. On the other hand, Iter does cover cases that our rather restricted and specialized Th. 3.5 does not – w.r.t. coverage, Iter seems to have a large overlap with the more widely applicable weak HOAS principle from Prop. 3.4 (the latter also appearing

[8] We restrict the discussion to iteration, in order to compare with our principle. In op. cit., one actually defines a recursor, not merely an iterator, and for the abstraction case one normalizes using funt for the "full-recursive" argument from $\mathbf{term}_d \to \mathbf{term}_d$.

in other places, as discussed above). [27] also discusses an induction principle following a similar line with Iter: the abstraction case considers arbitrary functions in $\mathbf{term}_d \to \mathbf{term}_d$, and uses their tbody in the induction hypothesis.

Built-in structural rules. The discussion of structural-like properties reveals a limitation of our HOAS approach compared to others. In HOAS-tailored approaches, provided the represented system is similar to the logical framework, the proofs of some of its properties becomes simple (some, e.g., weakening, even following directly from adequacy).

For instance, consider our running examples, β-reduction and the typing system of Curry-style System F (from Sections 3.2) and consider the task of proving type preservation for it:
- $\Gamma \vdash X : tX$ and $X \rightsquigarrow X'$ implies $\Gamma \vdash X' : tX$.
Regardless of the representation, the proof needs to go by induction on $X \rightsquigarrow X'$, and uses inversion rules for $\vdash$. For the inductive case of (Beta) being the last applied rule, a FOAS proof would require the following substitution lemma, henceforth called SL (see, e.g., [19]):
- If $\Gamma, x : tX, \Gamma' \vdash Y : tY$ and $\Gamma \vdash X : tX$, then $\Gamma, \Gamma' \vdash Y[X/x] : tY$.
(Note that SL is really a combination between a substitutivity and a cut rule.)

In a Twelf encoding (and similarly in other settings, including two-level Hybrid), one can dispense of SL, handling it simply by instantiating a universally quantified variable and by function application.[9] In our "one-level" setting, since we are really extracting HOAS from the implicit meta-language of classical mathematics (as opposed to an explicit intuitionistic language, as in Twelf and two-level Hybrid), we would still need to prove the "cut" part of SL, obtaining only the substitutivity part for free. Namely, we would need to prove the following (about our HOAS relation $\vdash$):
- If $\Delta, X : tX, \Delta' \vdash Y : tY$ and $\Delta \vdash X : tX$, then $\Delta, \Delta' \vdash Y : tY$.
(The proof goes through by trivial induction.) Note also that neither for this property, nor for type-preservation, can we employ the context-free induction principles from Ths. 3.9 and 3.10, since the conclusion of the fact to be proved involves contexts. Moreover, a clause such as ($\mathrm{Arr}\mathrm{I}_\theta$) from these theorems cannot be used as an inversion rule, as with traditional inductive definitions.

3.7.2 Strong normalization proofs for System F

The first proof of strong normalization for System F was given in Girard's Ph.D. thesis [53], the very place where (a Church-typed version of) the system was introduced. All the proofs that followed employed in one way or another Girard's original idea of *reducibility candidates*, in later papers by different authors called (under slightly different technical

[9] See also [112] for an illustration of another such LF-specific proof shortcut in the context of the POPLmark challenge [5].

conditions) *saturated sets* – Section 11 in [51] gives an overview. Variations in these proofs include the employment of terms that may or may not bear type annotations and technical adjustments on the "candidates". Our own proof follows Girard's idea as well, but brings a twofold improvement over previous proofs:
(1) It delves more directly into the heart of the problem – our *general-purpose* HOAS induction principle[10] expressed by Th. 3.10 "invites" one to seek a notion of candidate.
(2) It does away with the notions of *typing context*, and *type* or *data environment*, which are employed in *all* the previous proofs as "auxiliaries" to the main proof idea. Indeed, previous proofs define a variant of our type evaluation map θ (required to apply Th. 3.10) that is *parameterized by type environments*, i.e., by maps from tvars to tterms. Instead, we employ our compositionality criterion (Th. 3.5) to obtain a lightweight, non-parameterized θ directly, verifying what is known as "Girard's trick" (namely, proving that it has its image in the set of candidates) in a more transparent fashion. Then, previous proofs define a notion of semantic deduction in contexts, universally quantifying over type environments and/or data environments, and prove the typing relation sound w.r.t. it – this step is *not* required by our proof; more precisely, this routine issue of logical soundness has been recognized as a general phenomenon pertaining to HOAS and has already been dealt with in the proof of Th. 3.5.

On the formalization side, we are only aware of the LEGO [3] formalization from [12], and of the ATS [29] formalization from [41], both following [54]. The former uses de Bruijn encoding of the syntax, while the latter employs LF-style, axiomatic HOAS for data terms and de Bruijn indices for type terms. It appears that potential ATS variants of some of our results (mainly Ths. 3.9, 3.10 and 3.5) could have been used to "HOASify" (and simplify) the proof from [41] – in particular, our employment of Th. 3.5 seems to answer the following question raised in op. cit., on page 120: "[can one] prove strong normalization using a higher-order representation for types[?]". On the other hand, due to the partly axiomatic approach, the adequacy of the HOAS representation from op. cit. (i.e., variants of our Props. 3.6 and 3.8) cannot be formally established in that setting.

[10] "General-purpose", in that it is *not* an ad hoc principle aimed at proving the particular strong normalization result, but a general one derived by mere syntactic analysis of the typing system; analogous principles are possible for a large class of typing systems.

Chapter 4

Process algebra [1]

4.1 Introduction

This chapter contains the second part of this dissertation's contribution: a formalized incremental coinductive proof system for bisimilarity in process algebra.

Bisimilarity is arguably the most natural equivalence on interactive processes. Assuming process transitions are labeled by (observable) actions a, processes P and P' are bisimilar iff: **(I)** whenever P can a-transit to a process Q, P' can also a-transit to some process Q' such that P' and Q' are again bisimilar; **(II)** and vice versa; **(III)** and so on, *indefinitely* (as in an infinite game).

The above informal description of the bisimilarity relation can of course be made rigorous by defining bisimilarity to be the largest *bisimulation*, i.e., the largest relation θ for which (I) and (II) hold (with "bisimilar" replaced by "in θ"). But the largest-fixpoint description loses (at least superficially) the game-theoretic flavor of the intuitive description, so we stick to the latter for a while. How would one go about proving that P and Q are bisimilar? Well, if one were allowed an infinite proof, one could try to show that each transition of P is matched by a transition of Q so that the continuations P' and Q' are (claimed to be) bisimilar (and vice versa), and then prove the bisimilarity claims about all pairs of continuations P' and Q', and so on. This way, one would build an infinite tree whose nodes contain bisimilarity claims about pairs of processes. Now assume that, while expanding the tree, one encounters a repetition of a previous claim (that appeared on an ancestor node). A reasonable "optimization" of the infinite proof would then be to stop and "seal" that node, because the bisimilarity argument for its ancestor can be repeated ad litteram. In other words, one may take the (yet unresolved!) goal of the ancestor as a hypothesis, which discharges the repetitive goal – this is the upside of trying to build an infinite proof: non-well-foundedness (i.e., circularity) works in our advantage. Assume now one finds such repetitions on all paths when building the tree. Then our bisimilarity proof is done! In terms of the fixpoint definition, we have proved that the pair (P, Q) of processes located at the root are bisimilar by *coinduction*, i.e., by exhibiting a bisimulation that contains (P, Q). In

terms of proof engineering however, the needed bisimulation did not appear out of nowhere, but was built incrementally from the goal, essentially by an exploration that discovered a regular pattern for an infinite proof tree. In fact, *coinductive proofs are intuitively all about discovering regular patterns.*

This chapter provides formal support for this intuition. Here is an illustration of our approach, for a mini process calculus. Fix a set of actions **act** with a given silent action $\tau \in \mathbf{act}$ and a map on $\mathbf{act} \setminus \{\tau\}$, $a \mapsto \overline{a}$, such that $\overline{\overline{a}} = a$ for all $a \in \mathbf{act}$. The processes P are generated by the grammar:

$$P \ ::= \ 0 \mid a.P \mid P|Q \mid !P$$

Thus, we have idle process, action prefix, parallel composition, and replication. "!" binds more strongly than "|". The behavior of processes is specified by the following labeled transition system:

$$\frac{\cdot}{a.P \overset{a}{\rightsquigarrow} P}(\mathsf{PREF}) \qquad \frac{P_0 \overset{a}{\rightsquigarrow} Q_0}{P_0|P_1 \overset{a}{\rightsquigarrow} Q_0|P_1}(\mathsf{PARL}) \qquad \frac{P_1 \overset{a}{\rightsquigarrow} Q_1}{P_0|P_1 \overset{a}{\rightsquigarrow} P_0|Q_1}(\mathsf{PARR})$$

$$\frac{P_0 \overset{a}{\rightsquigarrow} Q_0 \quad P_1 \overset{\overline{a}}{\rightsquigarrow} Q_1}{P_0|P_1 \overset{\tau}{\rightsquigarrow} Q_0|Q_1}(\mathsf{PARS}) \qquad \frac{P \overset{a}{\rightsquigarrow} Q}{!P \overset{a}{\rightsquigarrow} !P|Q}(\mathsf{REPL}) \qquad \frac{P \overset{a}{\rightsquigarrow} Q_0 \quad P \overset{\overline{a}}{\rightsquigarrow} Q_1}{!P \overset{\tau}{\rightsquigarrow} !P|(Q_0|Q_1)}(\mathsf{REPLS})$$

We may wish to prove in this context that parallel composition is associative and commutative and that replication absorbs self-parallel composition, i.e., that

- $(P_0|P_1)|P_2 = P_0|(P_1|P_2)$,
- $P_0|P_1 = P_1|P_0$, and
- $P|!P = !P$

for all processes P_0, P_1, P_2, P, where we write "=" for *strong bisimilarity.* In fact, assume we already proved the first two facts and are left with proving the third, $P|!P = !P$. For this, we first check to see if the equations we already know so far (associativity and commutativity of $|$) imply this new one by pure equational reasoning – no, they don't. This means we cannot discharge the goal right away, and therefore we need to perform *unfoldings* of the two terms in the goal. We unfold $P|!P$ and $!P$ until we reach hypotheses involving only process meta-variables. The upper side of Figure 4.1 contains all possible *derived rules* (i.e., compositions of primitive rules in the system, all the way down to non-decomposable hypotheses) that can be matched by $P|!P$ in order to infer a transition from $P|!P$. And, similarly, the lower side for the term $!P$ – in this latter case, the matched derived rules coincide with the matched primitive rules. To see how the derived rules are obtained, the figure shows whole *derivation trees*, but we only care about the leaves and the roots of these trees.

Next, we try to pair these derived rules (upper versus lower), by the accordance of their hypotheses and their transition labels. The only valid pairing possibilities are: (1) with (5),

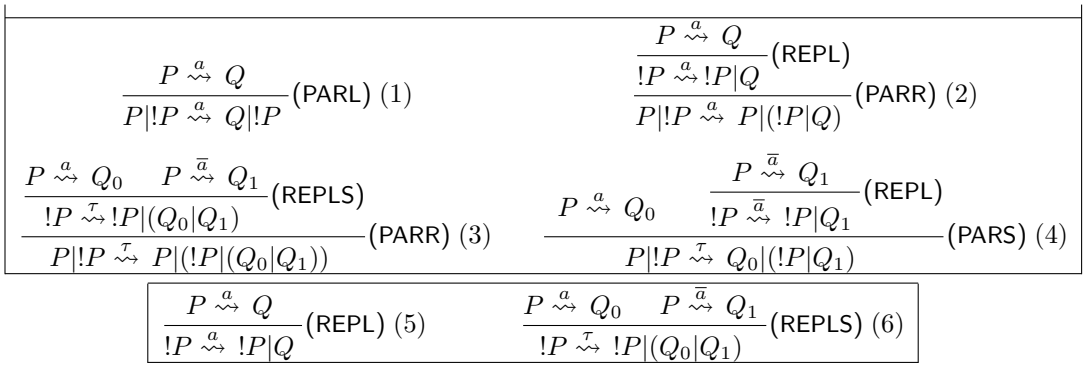

Figure 4.1: The matching derived rules for $P|!P$ and $!P$

(2) with (5), (3) with (6), and (4) with (6). The targets of the conclusions of the rules in these pairs yield four new goals:

- $Q|!P = !P|Q$;
- $P|(!P|Q) = !P|Q$;
- $P|(!P|(Q_0|Q_1)) = !P|(Q_0|Q_1)$;
- $Q_0|(!P|Q_1) = !P|(Q_0|Q_1)$.

The original goal, $P|!P = !P$, is replaced by the above four goals, *and is also henceforth taken as a hypothesis*. Notice that our goals are generic, i.e., universally quantified over the occurring process meta-variables, P, Q, Q_0, Q_1. Now, *equational reasoning* (by standard equational rules, *including substitution*) with hypothesis $P|!P = !P$ together with the already known lemmas $(P_0|P_1)|P_2 = P_0|(P_1|P_2)$ and $P_0|P_1 = P_1|P_0$ is easily seen to discharge each of the remaining four goals, and the proof is done.

Why is this proof valid, i.e., why does it represent a proof of the fact that, for all process terms P, $P|!P$ and $!P$ are bisimilar? The rigorous justification for this is the topic of this chapter. But the short answer has to do with our previous discussion on discovering patterns: the above is really a proof by coinduction (on universally quantified equalities of terms up to equational closure), which *builds incrementally* the relation representing the coinductive argument. Notice the appearance of *circular reasoning*: a goal that cannot be locally discharged is expanded according to the SOS definition transition relation and *becomes a hypothesis*. In this particular example, the proof is finished after only one expansion, but the process of expanding the goals with taking them as hypotheses may in principle continue, obtaining arbitrarily large proof trees.

We show that deductions such as the above are sound for a wide class of process algebras – those specifiable by SOS rules in the de Simone format [37]. Our results have been given a formalization in Isabelle/HOL [103], with the potential of leading to a (certified) implementation of a coinductive tool.

Here is an outline of this chapter. In Section 4.2, we discusses a representation of the de Simone format. Sections 4.3 and 4.4 contain our original theoretic contribution: incremental proof systems for bisimilarity – Section 4.3 for standard bisimilarity, Section

4.4 for universally quantified bisimilarity equations. In Section 4.5, we show how recursion and weak bisimilarity fall in the scope of our results. In Section 4.6, we give more technical examples illustrating our results. As usual, discussions of formalization and related work, in Sections 4.7 and 4.8, end the chapter.

4.2 Syntax and operational semantics of processes

Process variables, terms and substitution. We fix the following sets: **param**, of *parameters*, ranged over by p; **opsym**, of *operation symbols* (*opsyms* for short), ranged over by f, g; **var**, of *(process) variables*, ranged over by X, Y, Z – this latter set is assumed to be infinite. The set **term**, of *(process) terms*, ranged over by P, Q, R, T, S, U, V, is defined as follows, where ps and Ps range over lists of parameters and lists of terms, as follows:

$$P \quad ::= \quad \mathsf{Var}\ X \mid \mathsf{Op}\ f\ ps\ Ps$$

Thus, a term can have any opsym at the top, applied to any list of parameters and any list of terms (of any length), without being subject to further well-formedness conditions. Hence an opsym f does *not* have an a priori associated numeric rank (m, n) (indicating that f takes m parameters and n terms). Rather, we allow in **term** the whole pool of all possible terms under all possible rankings of the operation symbols. This looseness w.r.t. terms is admittedly a formalization shortcut (fitting nicely the Isabelle simply-typed framework), but is completely unproblematic for the concepts and results of this chapter: while an SOS specification of a transition system will of course employ only certain (possibly overloaded) ranks for the opsyms, the unused ranks will be harmless, since they will not affect transitions or bisimilarity.

σ and τ will range over $\mathbf{var} \to \mathbf{term}$. We consider the operators:
- $\mathsf{vars} : \mathbf{term} \to \mathbf{P}(\mathbf{var})$, giving the set of variables occurring in a term.
- $\_[\_] : \mathbf{term} \to (\mathbf{var} \to \mathbf{term}) \to \mathbf{term}$, such that (as usual) $T[\sigma]$ is the term obtained from T by substituting all its variables X by $\sigma\, X$.

Next we represent the meta-SOS notion of a *transition-system specification* [59, 97]. Given any set A, the set $\mathbf{Ftrans}(A)$, of formal A-transitions, consists of pairs, written $k \rightsquigarrow l$, with $k, l \in A$, where k is called the *source* and l the *target*. We fix a set **act**, of *actions*, ranged over by a, b.

Rules syntax. The set **rule**, of *(SOS-)rules*, ranged over by rl, is defined consist of triples $(\mathsf{hyps}, \mathsf{cnc}, \mathsf{side})$, where:
- $\mathsf{hyps} \in \mathbf{List}((\mathbf{Ftrans}(\mathbf{var})))$, read "hypotheses";
- $\mathsf{cnc} \in \mathbf{Ftrans}(\mathbf{term})$, read "conclusion";

- side : $(I\!N \to \mathbf{act}) \to \mathbf{act} \to \mathbf{bool}$, read "side-condition".

The hypotheses and the conclusions of our rules are therefore formal transitions between variables, and between terms, respectively. I.e., for any rule rl:
- hyps rl has the form $[XX_0 \rightsquigarrow Y_0, \ldots, XX_{n-1} \rightsquigarrow Y_{n-1}]$, with XX_j, Y_j variables;
- cnc rl has the form $S \rightsquigarrow T$, with S and T terms.

One can visualize rl as

$$\frac{XX_0 \rightsquigarrow Y_0, \ldots, XX_{n-1} \rightsquigarrow Y_{n-1}}{S \rightsquigarrow T}\ [\lambda\, as, b.\ \mathsf{side}\ rl\ as\ b]$$

where $as : I\!N \to \mathbf{act}$ and $b : \mathbf{act}$. Actually, we think of rl as follows:

$$\frac{XX_0 \overset{as\ 0}{\rightsquigarrow} Y_0\ , \ldots, XX_{n-1} \overset{as\ (n-1)}{\rightsquigarrow} Y_{n-1}}{S \overset{b}{\rightsquigarrow} T}\ [\mathsf{side}\ rl\ as\ b]$$

Note however that the side condition side rl is (for now) allowed to take into consideration the whole function as, and not only its first n values $as\ 0, \ldots, as\ (n-1)$, as one would expect – this is corrected below by "saneness".

Given a rule rl with hyps rl and cnc rl as above, we write:
- theXXs rl, for the variable list $[XX_0, \ldots, XX_{n-1}]$;
- theYs rl, for the variable list $[Y_0, \ldots, Y_{n-1}]$;
- theS rl, for the term S;
- theT rl, for the term T.

A rule rl is said to be *sane* if the following hold:
- **(1)** theYs rl is nonrepetitive;
- **(2)** set(theXXs rl) $\subseteq$ vars(theS rl);
- **(3)** vars(theS rl) $\cap$ set(theYs rl) $= \emptyset$;
- **(4)** vars(theT rl) $\subseteq$ vars(theS rl) $\cup$ set(theYs rl);
- **(5)** $\forall as, as'.\ (\forall i < \mathsf{length}(\mathsf{theYs}\ rl).\ as\ i = as'\ i) \Longrightarrow \mathsf{side}\ rl\ as = \mathsf{side}\ rl\ as'$.

A rule rl is said to be *amenable* if theS rl has the form Op f ps [Var $X_0, \ldots,$ Var X_{m-1}], where f is an opsym, ps a list of parameters, and $[X_0, \ldots, X_{m-1}]$ a *nonrepetitive* list of variables. Given an amenable rule rl as above, we write:
- thef rl for f,
- theps rl for ps,
- theXs rl for $[X_0, \ldots, X_{m-1}]$.

Saneness expresses a natural property for well-behaved SOS rules: Think of a term S as a generic composite process, built from its unspecified components (its variables) by means of opsyms. Then a sane rule is one that describes the behavior of the composite S in terms of the behavior of (some of) its components: condition (2) says that indeed the hypotheses refer to the components, (1) and (3) that the hypotheses only assume that some

components transit "somewhere" (without any further information), (4) that the resulted continuation of the composite depends only on the components and their continuations, and (5) that the side-condition may only depend on the action labels of the hypotheses and of the conclusion. In addition, amenability asks that the composite process S be obtained by a primitive operation f applied to unspecified components. The conjunction of saneness and amenability is precisely the de Simone format requirement [37], hence we call a rule *de Simone* if it is sane and amenable.

Running example. We show what the example in the introduction becomes under our representation. Assume that **act** is an unspecified set with constants $^- : \mathbf{act} \to \mathbf{act}$ and $\tau \in \mathbf{act}$ such that $\overline{\overline{a}} = a$ for all $a \neq \tau$. Define the relation $\mathsf{sync} : \mathbf{act} \to \mathbf{act} \to \mathbf{act} \to \mathbf{bool}$ by $\mathsf{sync}\ a\ b\ c = (a \neq \tau \wedge b \neq \tau \wedge \overline{a} = b \wedge c = \tau)$. We take **opsym** to be a three-element set $\{\mathsf{Pref}, \mathsf{Par}, \mathsf{Repl}\}$ and **param** to be **act**. For readability, in our running example (including throughout the future continuations of this example), for all $X \in \mathbf{var}$, $S, T \in \mathbf{term}$ and $a \in \mathbf{act}$, we use the following abbreviations:

- X for $\mathsf{Var} \in X$;
- $a.S$ for $\mathsf{Op\ Pref}\ [a]\ [S]$;
- $S \mid T$ for $\mathsf{Op\ Par\ Nil}\ [T, S]$;
- $!\,S$ for $\mathsf{Op\ Repl\ Nil}\ [S]$.

Rls consists of the rules $\{\mathsf{PREF}_a.\ a \in \mathbf{act}\} \cup \{\mathsf{PARL}, \mathsf{PARR}, \mathsf{PARS}, \mathsf{REPL}, \mathsf{REPLS}\}$ listed below, where X, Y, X_0, X_1, Y_0, Y_1 are fixed *distinct* variables.

$$\frac{\cdot}{a.X \overset{b}{\rightsquigarrow} X}\ [a = b]\ (\mathsf{PREF}_a) \qquad \frac{X_0 \overset{as\,0}{\rightsquigarrow} Y_0}{X_0 \mid X_1 \overset{b}{\rightsquigarrow} Y_0 \mid X_1}\ [as\,0 = b]\ (\mathsf{PARL})$$

$$\frac{X_0 \overset{as\,0}{\rightsquigarrow} Y_0}{X_1 \mid X_0 \overset{b}{\rightsquigarrow} X_1 \mid Y_0}\ [as\,0 = b]\ (\mathsf{PARR}) \qquad \frac{X_0 \overset{as\,0}{\rightsquigarrow} Y_0 \quad X_1 \overset{as\,1}{\rightsquigarrow} Y_1}{X_0 \mid X_1 \overset{b}{\rightsquigarrow} Y_0 \mid Y_1}\ [\mathsf{sync}\,(as\,0)\,(as\,1)\,b]\ (\mathsf{PARS})$$

$$\frac{X \overset{as\,0}{\rightsquigarrow} Y}{!X \overset{b}{\rightsquigarrow} !X \mid Y}\ [as\,0 = b]\ (\mathsf{REPL}) \qquad \frac{X \overset{as\,0}{\rightsquigarrow} Y_0 \quad X \overset{as\,1}{\rightsquigarrow} Y_1}{!X \overset{b}{\rightsquigarrow} !X \mid (Y_0 \mid Y_1)}\ [\mathsf{sync}\,(as\,0)\,(as\,1)\,b]\ (\mathsf{REPLS})$$

For listing the rules, we employed the previously discussed visual representation. E.g., the formal description of PARS is $(\!|\ \mathsf{hyps} = [X_0 \rightsquigarrow Y_0,\ X_1 \rightsquigarrow Y_1];\ \mathsf{cnc} = (X_0 \mid X_1 \rightsquigarrow Y_0 \mid Y_1);\ \mathsf{side} = (\lambda\ as, b.\ \mathsf{sync}\,(as\,0)\,(as\,1)\,b)\ |\!)$. All the rules in this example are easily seen to be de Simone.

Rules semantics. We fix *Rls*, a set of de Simone rules. The one-step transition relation induced by *Rls* on terms is a (curried) ternary relation $\mathsf{step} : \mathbf{term} \to \mathbf{act} \to \mathbf{term} \to \mathbf{bool}$, where we write $P \overset{a}{\rightsquigarrow} Q$ instead of $\mathsf{step}\ P\ a\ Q$, defined inductively by the following clause:

- if $rl \in Rls$, $\sigma((\mathsf{theXXs}\ rl)!j) \overset{as\,j}{\rightsquigarrow} \sigma((\mathsf{theYs}\ rl)!j)$ for all $j < \mathsf{length}(\mathsf{theYs}\ rl)$, and $\mathsf{side}\ rl\ as\ b$ holds, then $(\mathsf{theS}\ rl)[\sigma] \overset{b}{\rightsquigarrow} (\mathsf{theT}\ rl)[\sigma]$

(where $\sigma : \mathbf{var} \to \mathbf{term}$, $as : I\!N \to \mathbf{act}$, and $b \in \mathbf{act}$).

The above definition is the expected one: each (generic) rule in *Rls* yields, for each substitution of the variables in the rule and for each choice of the actions fulfilling the side-condition, an inference of the instance of the rule's conclusion from the instances of the rule's hypotheses.

Bisimilarity. We write **rel** for $\mathbf{P}(\mathbf{term} \times \mathbf{term})$, the set of relations between terms, ranged over by θ, η, ξ. The (monotonic) *retract* operator $\mathsf{Retr} : \mathbf{rel} \to \mathbf{rel}$, named so because it maps each θ to a relation retracted (w.r.t. transitions) back from θ, is defined by:
- $\mathsf{Retr}\ \theta = \{(P,Q).\ (\forall a, P'.\ P \overset{a}{\rightsquigarrow} P' \implies (\exists Q'.\ (P',Q') \in \theta \wedge Q \overset{a}{\rightsquigarrow} Q'\ )) \wedge (\forall a, Q'.\ Q \overset{a}{\rightsquigarrow} Q' \implies (\exists P'.\ (P',Q') \in \theta \wedge P \overset{a}{\rightsquigarrow} P'\ ))\}$.
The *bisimilarity* relation, $\mathsf{bis} \in \mathbf{rel}$, is the *greatest fixed point* of Retr.

Notice that we defined bisimilarity for *open* terms (i.e., terms possibly containing variables), while often in the literature both transition and bisimilarity are defined for *closed* terms only (with step and Retr defined by the same conditions as above, but acting on closed terms and relations on closed terms, respectively). However, for the de Simone format of our rules (as well as for more general formats, e.g., well-founded pure tyft [59]), transition does *not* bring any variables (in the sense that, if $P \overset{a}{\rightsquigarrow} P'$, then the free variables of P are among those of P') implying that two closed terms are bisimilar according to our definition iff they are bisimilar according to the aforementioned "closed" version.

Because of the particular format of the rules, bis is a congruence on terms. This is in fact true for rule formats more expressive than the one considered here [24, 59, 135]. However, we shall need to exploit a stronger property specific to the de Simone format, namely: whenever θ is a congruence, it follows that $\theta \cap (\mathsf{Retr}\ \theta)$ is also a congruence. Let, for any relation θ, $\mathsf{congCl}\ \theta$ be its congruence closure. From the above, we infer a powerful "up to" coinduction rule (that is, up to bisimilarity and up to arbitrary contexts), due to de Simone [37] and Sangiorgi [137], improving on traditional coinduction:

Theorem 4.1 *For all* $\theta \in \mathbf{rel}$, *if* $\theta \subseteq \mathsf{Retr}(\mathsf{congCl}(\theta \cup \mathsf{bis}))$, *then* $\theta \subseteq \mathsf{bis}$.

4.3 The raw coinductive proof system

We now present the core of our original theoretical contribution: defining an incrementally-coinductive proof system for bisimilarity and proving it sound. We define the *raw deduction* relation $\vdash : \mathbf{rel} \to \mathbf{rel} \to \mathbf{bool}$ (with infix notation) inductively by the clauses:

$$\frac{\cdot}{\theta \vdash \theta'}\ (\mathsf{Ax})\ [\theta' \subseteq \mathsf{congCl}(\theta \cup \mathsf{bis})] \qquad \frac{\forall \theta' \in \Theta.\ \theta \vdash \theta'}{\theta \vdash \bigcup \Theta}\ (\mathsf{Split})\ [\Theta \neq \emptyset] \qquad \frac{\theta' \cup \theta \vdash \theta''}{\theta \vdash \theta'}\ (\mathsf{Coind})\ [\theta' \subseteq \mathsf{Retr}\ \theta'']$$

$\theta \vdash \theta'$ is eventually intended to mean: "θ implies θ' modulo bisimilarity and congruence closure". Here is the intuitive reading of the rules (thinking of them as being applied backwards for expanding or discharging goals). (Ax) allows to deduce θ' from θ right away. (Split) allows for splitting the goal according to a chosen partition of its conclusion. (Coind) is the interesting rule, and is the actual engine of the proof system. To get an intuitive grasp of this rule, let us first assume that $\theta = \emptyset$ (i.e., that θ is empty). Then the goal is to show θ' included in $\mathsf{congCl\ bis}$, i.e., in bis. For this, it would suffice that $\theta' \subseteq \mathsf{Retr}(\theta')$; alternatively, we may "defer" the goal by coming up with an "interpolant" θ'' such that $\theta' \subseteq \mathsf{Retr}(\theta'')$ and θ' implies θ'' modulo bisimilarity and congruence. (As we shall see in the next section, working symbolically with open terms provides natural interpolant candidates.) In case $\theta \neq \emptyset$, θ should be thought of *temporally* as the collection of auxiliary facts gathered from previous coinductive expansions.

Note that, for the aforementioned intention of the proof system, (Coind) is not sound *by itself*: regarded as applied backwards to a goal, it moves the conclusion θ' to the hypotheses, creating a circularity. In other words, of course it is not true that the conjunction of $\theta'' \subseteq \mathsf{congCl}(\theta' \cup \theta \cup \mathsf{bis})$ and $\theta' \subseteq \mathsf{Retr}\ \theta''$ implies $\theta' \subseteq \mathsf{congCl}(\theta \cup \mathsf{bis})$ for all $\theta, \theta', \theta''$. Yet, *the proof system as a whole* is sound in the following sense:

Theorem 4.2 *If $\emptyset \vdash \theta$, then $\theta \subseteq \mathsf{bis}$.*

In the remainder of this section, we outline the proof of this theorem.

(I) In order to gain more control on the proof system, we *objectify* it in a standard fashion, by considering proofs (i.e., proof trees) explicitly, at the object level (as opposed to merely implicitly as they appear in the inductive definition of $\vdash$). For this, we pick a sufficiently large set **index**, ranged over by i, and define the set **prf**, of *proof trees*, ranged over by Pf, with constructors mirroring the clauses in the definition of $\vdash$, as follows (where Pfs ranges over $\mathbf{index} \to \mathbf{prf}$ and θ, θ' over **rel**):

$$Pf \quad ::= \quad \mathsf{Ax}\ \theta\ \theta' \mid \mathsf{Split}\ Pfs\ \theta\ \theta' \mid \mathsf{Coind}\ Pf\ \theta\ \theta'$$

The pair of relations that a proof tree Pf "proves", which is (θ, θ') when Pf has the one of the forms $\mathsf{Ax}\ \theta\ \theta'$, $\mathsf{Split}\ Pfs\ \theta\ \theta'$, or $\mathsf{Coind}\ Pf\ \theta\ \theta'$, is denoted by $\mathsf{proves}\ Pf$. The conclusion-hypothesis dependencies and the side-conditions of the clauses defining $\vdash$ are captured by the predicate $\mathsf{correct} : \mathbf{prf} \to \mathbf{bool}$, defined recursively as expected:

- $\mathsf{correct}\ (\mathsf{Ax}\ \theta\ \theta') = (\theta' \subseteq \mathsf{congCl}(\theta \cup \mathsf{bis}))$;
- $\mathsf{correct}\ (\mathsf{Split}\ Pfs\ \theta\ \theta') = ((\forall i.\ \mathsf{correct}(Pfs\ i) \wedge \mathsf{fst}(\mathsf{proves}(Pfs\ i)) = \theta) \wedge \bigcup i.\ \mathsf{snd}(\mathsf{proves}(Pfs\ i)) = \theta')$;
- $\mathsf{correct}\ (\mathsf{Coind}\ Pf\ \theta\ \theta') = (\mathsf{correct}\ Pf \wedge \mathsf{fst}(\mathsf{proves}\ Pf) = \theta' \cup \theta \wedge \theta' \subseteq \mathsf{Retr}(\mathsf{snd}(\mathsf{proves}\ Pf)))$.

It is immediate that $\theta \vdash \theta'$ holds iff $\exists Pf.\ \mathsf{correct}(Pf) \wedge \mathsf{proves}(Pf) = (\theta, \theta')$.

(II) Thus, it suffices to show that $\theta \subseteq \mathsf{bis}$ whenever there exists a correct proof tree Pf such that $\mathsf{proves}(Pf) = (\theta, \theta')$. For showing the latter, we introduce a couple of auxiliary concepts. Given Pf, a *label* in Pf is a pair (θ, θ') "appearing" in Pf – formally, we define $\mathsf{labels} : \mathbf{prf} \to \mathbf{P}(\mathbf{rel} \times \mathbf{rel})$ by:

- $\mathsf{labels}\,(\mathsf{Ax}\ \theta\ \theta') = \{(\theta, \theta')\}$;
- $\mathsf{labels}\,(\mathsf{Split}\ Pfs\ \theta\ \theta') = \{(\theta, \theta')\} \cup \bigcup i.\ \mathsf{labels}(Pfs\ i)$;
- $\mathsf{labels}\,(\mathsf{Coind}\ Pf\ \theta\ \theta') = \{(\theta, \theta')\} \cup \mathsf{labels}\ Pf$.

We let $\mathsf{Left}\ Pf$ denote the union of the lefthand sides of all labels in Pf, and $\mathsf{Right}\ Pf$ the union of the righthand sides of all labels in Pf.

Lemma 4.3 *If Pf is correct, then* $\mathsf{Right}\ Pf \subseteq \mathsf{congCl}((\mathsf{Left}\ Pf) \cup \mathsf{bis})$.

Lemma 4.4 *If Pf is correct and* $\mathsf{fst}(\mathsf{proves}\ Pf) \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf)$, *then* $\mathsf{Left}\ Pf \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf)$.

Lemma 4.3 follows by an easy induction on proof trees. By contrast, Lemma 4.4 requires some elaboration – before getting into that, let us show how the two lemmas imply our desired fact. Assume that Pf is correct and $\mathsf{proves}\ Pf = (\emptyset, \theta)$. Then the hypotheses of both lemmas are satisfied by Pf, and therefore (since also Retr is monotonic) $\mathsf{Left}\ Pf \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf) \subseteq \mathsf{Retr}(\mathsf{congCl}((\mathsf{Left}\ Pf) \cup \mathsf{bis}))$, implying, by Theorem 4.1, $\mathsf{Left}\ Pf \subseteq \mathsf{bis}$. With Lemma 4.3, we obtain $\mathsf{Right}\ Pf \subseteq \mathsf{congCl}(\mathsf{bis})$, which means (given that bis is a congruence) $\mathsf{Right}\ Pf \subseteq \mathsf{bis}$. And since $\theta \subseteq \mathsf{Right}\ Pf$, we obtain $\theta \subseteq \mathsf{bis}$, as desired.

It remains to prove Lemma 4.4. This lemma states a property of proof trees that depends on a hypothesis concerning their roots (i.e., the pair (θ, θ') that they "prove"). The task of finding a strengthening of that hypothesis so that a direct proof by structural induction goes through seems rather difficult, if not impossible. We instead take the roundabout route of identifying an invariant satisfied on backwards paths in the proof trees whose roots satisfy our hypothesis. First, we define the notion of a *path* (independently of proof trees): a list $[(\theta_0, \theta'_0), \ldots, (\theta_{m-1}, \theta'_{m-1})]$ is called a *path* if the following is true for all $n < m - 1$: either $\theta_{n+1} = \theta_n$, or $\theta_{n+1} \subseteq \mathsf{Retr}(\theta'_{n+1}) \cup \theta_n$. Then one can verify the following:

-(a) Fix $\xi \in \mathbf{rel}$. If $[(\theta_0, \theta'_0), \ldots, (\theta_{m-1}, \theta'_{m-1})]$ is a path, $\theta_0 \subseteq \mathsf{Retr}\ \xi$ and $\forall n < m.\, \theta'_n \subseteq \xi$, then $\forall n < m.\, \theta_n \subseteq \mathsf{Retr}\ \xi$. (By easy induction on n.)

-(b) If Pf is correct, $\mathsf{proves}(Pf) = (\theta, \theta')$, and (η, η') is a label in Pf, then there exists a path $[(\theta_0, \theta'_0), \ldots, (\theta_{m-1}, \theta'_{m-1})]$ consisting of labels in Pf (i.e., such that (θ_n, θ'_n) are labels in Pf for all $n < m$) and connecting (θ, θ') with (η, η') (i.e., such that $(\theta_0, \theta'_0) = (\theta, \theta')$ and $(\theta_{m-1}, \theta'_{m-1}) = (\eta, \eta')$). (By induction on Pf.)

With these preparations, we can prove Lemma 4.4: Assume $\mathsf{proves}(Pf) = (\theta, \theta')$ and $\theta \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf)$. Fix a label (η, η') in Pf. According to (b), there exists a path connecting

(θ, θ') with (η, η') and going through labels in Pf only. Then the hypotheses of (a) are satisfied by the aforementioned path and $\xi = \mathsf{Right}\ Pf$, and therefore all the lefthand sides of the pairs in this path are included in $\mathsf{Retr}(\mathsf{Right}\ Pf)$. In particular, $\eta \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf)$. Since the choice of the label (η, η') was arbitrary, it follows that $\mathsf{Left}\ Pf \subseteq \mathsf{Retr}(\mathsf{Right}\ Pf)$, as desired.

Remarks. (1) The soundness of $\vdash$ was established not locally (rule-wise), as is customary in soundness results, but globally, by analyzing entire proof trees. What the potential backwards applications of the clause (Coind) do is to *improve the candidate relation for the coinductive argument*. In the end, as shown by the proof of Theorem 4.2, the (successful) relation is synthesized by putting together the righthand sides of all labels in the proof tree.

(2) The proof system represented by $\vdash$ is not a typical syntactic system, but contains semantic intrusions – in effect, the system is complete already by its axiom (Ax), which allows for an instantaneous "oracle proof" that the considered relation is included in bisimilarity. But of course, the realistic employment of this system will appeal to such instantaneous proofs only through the available (already proved) lemmas. (Thus, the purpose of including bis in the side-condition of (Ax) was not to ensure completeness (in such a trivial manner), but to allow the usage of previously known facts about bisimilarity.) A more syntactic and syntax-driven system for terms (also featuring oracles though, for the same reason as this one) will be presented in the next section.

4.4 Deduction of universally quantified bisimilarity equations

Next we introduce a deduction system for term equalities, where, as before, we interpret equality as bisimilarity, but now we interpret the occurring variables as being *universally quantified over the domain of terms*.

Universal bisimilarity, $\mathsf{ubis} \in \mathbf{rel}$, is defined as follows: $(U, U') \in \mathsf{ubis}$ iff $(U[\tau], U'[\tau]) \in \mathsf{bis}$ for all substitutions $\tau : \mathbf{var} \to \mathbf{term}$.

Thus, e.g., given distinct variables X and Y and an opsym f,
- $(\mathsf{Op}\ f\ \mathsf{Nil}\ [\mathsf{Var}\ X, \mathsf{Var}\ Y], \mathsf{Op}\ f\ \mathsf{Nil}\ [\mathsf{Var}\ Y, \mathsf{Var}\ X]) \in \mathsf{ubis}$

is equivalent to
- $\forall U, V \in \mathbf{term}.\ (\mathsf{Op}\ f\ \mathsf{Nil}\ [U, V], \mathsf{Op}\ f\ \mathsf{Nil}\ [V, U]) \in \mathsf{bis}$.

Matched derived rules. Derived rules appear by composing primitive rules (i.e., the de Simone rules in *Rls*) within maximal composition chains. I.e., they come from considering,

in the SOS system, derivation trees that are completely backwards-saturated (in that their leaves involve only variables as sources and targets) and then forgetting the intermediate steps in these trees. A derived rule may not be amenable (hence not de Simone), but will always be sane. We shall let *drl* denote derived rules, keeping the symbol *rl* for primitive rules.

We are interested in constructing all derived rules that are matched by a given term U in such a way that U becomes the source of the conclusion of the derived rule; in doing so, we also care about avoiding any overlap between the freshly generated variables (required to build the rules) and the variables of another given term V (that we later wish to prove universally bisimilar with U). We thus introduce the operator $\mathsf{mdr} : \mathbf{term} \to \mathbf{term} \to \mathbf{P}(\mathbf{rule})$, read "matched derived rules", such that, given $U, V \in \mathbf{term}$, $\mathsf{mdr}\ V\ U$ is the set of all the derived rules with U as the source of their conclusion and with "the Ys" fresh for V. We write $\mathsf{mdr}_V\ U$ instead of $\mathsf{mdr}\ V\ U$.

The definition of mdr is both intuitive and standard (and was already sketched in the pioneering paper [37]), but its formalities are very technical, due to the need to avoid name overlapping and compose side-conditions.[2] $\mathsf{mdr} : \mathbf{term} \to \mathbf{term} \to \mathbf{P}(\mathbf{rule})$ is defined recursively on the second argument as follows:

(I) $\mathsf{mdr}\ V\ (\mathsf{Var}\ X)$ consists of a single rule: $\dfrac{X \overset{as\,0}{\rightsquigarrow} Y}{\mathsf{Var}\ X \overset{b}{\rightsquigarrow} \mathsf{Var}\ Y}\ [b = as\,0]$, where Y is a choice of a variable fresh for X and V. (Thus, an "identity" rule.)

(II) $\mathsf{mdr}\ V\ (\mathsf{Op}\ f\ ps\ [U_0, \ldots, U_{m-1}])$ contains one rule (that we later refer to as "the promissed rule") for each $rl \in Rls$, $n \in I\!N$ and $[drl_0, \ldots, drl_{n-1}] \in \mathbf{List}(\mathbf{rule})$ satisfying the following two conditions:

— rl has the form $\dfrac{XX_0 \overset{as\,0}{\rightsquigarrow} Y_0\ ,\ \ldots\ , XX_{n-1} \overset{as\,(n-1)}{\rightsquigarrow} Y_{n-1}}{\mathsf{Op}\ f\ ps\ [\mathsf{Var}\ X_0, \ldots, \mathsf{Var}\ X_{m-1}] \overset{b}{\rightsquigarrow} T}\ [\mathsf{side}\ as\ b]$.
(Thus, rl has n hypotheses and the source of its conclusion has the opsym f and the parameter list ps at the top, and has precisely m immediate subterms which, by the de Simone format requirement, have to be distinct variables.)

— Given $\sigma : \mathbf{var} \to \mathbf{term}$ defined by $X_0 \mapsto U_0$, ..., $X_{m-1} \mapsto U_{m-1}$ (and with all the other variables mapped by σ no matter where), it holds that $drl_j \in \mathsf{mdr}\ V\ (\sigma\ XX_j)$ for all $j < n$. (Note also that $\sigma\ XX_j = U_i$, where i is the unique $k \in \{0, \ldots, m-1\}$ such that $X_k = XX_j$.)

Given rl and $[drl_0, \ldots, drl_{n-1}]$ as above, we construct the promised rule. Write drl_j as $\dfrac{XX^j_0 \overset{as^j\,0}{\rightsquigarrow} Y^j_0\ ,\ \ldots\ , XX^j_{k_j-1} \overset{as^j\,(k_j-1)}{\rightsquigarrow} Y^j_{k_j-1}}{S^j \overset{b^j}{\rightsquigarrow} T^j}\ [\mathsf{side}^j\ as^j\ b^j]$. We first perform (if necessary)

[2] In [24, 10], where what we call "matched derived rules" are called "ruloids", mdr is not even defined, but rather the existence of such an operator satisfying suitable properties (essentially our below soundness and completeness) is proved.

renamings of some of "the Ys" in the rules drl_j obtaining "copies" drl'_j verifying certain conditions (see below). Each drl'_j will have the form

$$\frac{XX^j_0 \overset{as^j\ 0}{\rightsquigarrow} Y'^j_0\ ,\ \ldots\ , XX^j_{k_j-1} \overset{as^j\ (k_j-1)}{\rightsquigarrow} Y'^j_{k_j-1}}{S^j \overset{b^j}{\rightsquigarrow} T'^j}\ [\mathsf{side}^j\ as^j\ b^j]$$

(Thus, "the XXs", "the S", and the side conditions do not change from drl_j to drl'_j.) The aforementioned conditions satisfied by drl'_j are the following:

-**(i)** for all $j < n$, drl'_j is also sane (like drl_j was);

-**(ii)** for all $j_1, j_2 < n$ with $j_1 \neq j_2$, $\mathsf{theYs}\ drl'_{j_1}$ is disjoint from $\mathsf{theYs}\ drl'_{j_2}$, from $\mathsf{vars}(\mathsf{theS}\ drl'_{j_2})$, and from $\mathsf{vars}\ V$;

-**(iii)** for all $j < n$, $T'^j = T^j[Y'^j_0/Y^j_0, \ldots, Y'_{k_j-1}{}^j/Y^j_{k_j-1}]$, where $Y'^j_0/Y^j_0, \ldots, Y'_{k_j-1}{}^j/Y^j_{k_j-1}$ is the map sending each Y'^j_l to $\mathsf{Var}\ Y_l{}^j$.

Now, let $\tau : \mathbf{var} \to \mathbf{term}$ update σ with $Y_0 \mapsto T'^0$, ..., $Y_{n-1} \mapsto T'^{n-1}$. Then our promised rule should be something like:

$$\frac{\begin{array}{lll} XX^0_0 \overset{as^0\ 0}{\rightsquigarrow} Y'^0_0\ , & \ldots & , XX^0_{k_0-1} \overset{as^0\ (k_0-1)}{\rightsquigarrow} Y'^0_{k_0-1} \\ & \vdots & \\ XX^{n-1}_0 \overset{as^{n-1}\ 0}{\rightsquigarrow} Y'^{n-1}_0\ , & \ldots & , XX^{n-1}_{k_{n-1}-1} \overset{as^{n-1}\ (k_{n-1}-1)}{\rightsquigarrow} Y'^{n-1}_{k_{n-1}-1} \end{array}}{\mathsf{Op}\ f\ ps\ [X_0, \ldots, X_{m-1}] \overset{b}{\rightsquigarrow} T[\tau]}\ [?]$$

thus having a number of $k_0 + k_1 + \ldots + k_{n-1}$ hypotheses. We have not indicated its side-condition yet. Intuitively, it should be the relational composition of side (the side-condition of rl) with the side^j-s (the side-conditions of the drl_j-s). This requires the standard linearization of the array of indexes $(j,l)_{j<n, l<k_j}$ into a list, mapping each (j,l) to $(\epsilon_j + l)$, where $\epsilon_j = \sum_{j'<j} k_{j'}$. Also, we need the operator $\mathsf{shift} : I\!N \to (I\!N \to \mathbf{act}) \to (I\!N \to \mathbf{act})$, defined by $\mathsf{shift}\ n\ as = \lambda\, i.\, as(n+i)$. Then the side-condition of our promised rule, call it pside, is defined as follows: $\mathsf{pside}\ as\ c = (\exists\ bs.\ \mathsf{side}\ bs\ c \wedge (\forall j < n.\ \mathsf{side}^j\ (\mathsf{shift}\ \epsilon_j\ as)\ (bs\, j)))$.

Our promised rule is therefore:

$(\!|\ \mathsf{hyps} = [XX^0_0 \rightsquigarrow Y'^0_0, \ldots, XX^0_{k_0-1} \rightsquigarrow Y'^0_{k_0-1},\ \ldots,\ XX^{n-1}_0 \rightsquigarrow Y'^{n-1}_0, \ldots, XX^{n-1}_{k_{n-1}-1} \rightsquigarrow Y'^{n-1}_{k_{n-1}-1}]$;

$\mathsf{cnc} = \mathsf{Op}\ f\ ps\ [X_0, \ldots, X_{m-1}] \rightsquigarrow T[\tau]$;

$\mathsf{side} = \mathsf{pside}\ |\!)$,

and we are finally done with the definition of mdr.

Running example (continued). We again assume that all the variables X, Y etc. that we refer to below are *fixed distinct variables*.

- $\mathsf{mdr}_{!X}(X \,|\, !X)$, the set of derived rules matched by $X \,|\, !X$ and with "the Ys" avoiding the variables of $!X$, consists of $\{\mathsf{DRL}_1, \mathsf{DRL}_2, \mathsf{DRL}_3, \mathsf{DRL}_4\}$ (see below);

- $\mathsf{mdr}_{X\,|\,!X}(!X)$, the set of derived rules matched by $!X$ and with "the Ys" avoiding the variables of $X \,|\, !X$, consists of $\{\mathsf{DRL}_5, \mathsf{DRL}_6\}$ (given below).

$$\frac{X \overset{as\,0}{\rightsquigarrow} Y}{X \mid !X \overset{b}{\rightsquigarrow} Y \mid !X}\ (\mathsf{DRL}_1)\ [as\ 0 = b] \qquad \frac{X \overset{as\,0}{\rightsquigarrow} Y}{X \mid !X \overset{b}{\rightsquigarrow} X \mid (!X \mid Y)}\ (\mathsf{DRL}_2)\ [\exists c.\ as\ 0 = c \wedge c = b]$$

$$\frac{X \overset{as\,0}{\rightsquigarrow} Y_0 \qquad X \overset{as\,1}{\rightsquigarrow} Y_1}{X \mid !X \overset{b}{\rightsquigarrow} X \mid (!X \mid (Y_0 \mid Y_1))}\ (\mathsf{DRL}_3)\ \left[\exists c.\ \begin{array}{l}\mathsf{sync}\,(as\,0)\,(as\,1)\,c \\ \wedge\ c = b\end{array}\right] \qquad \frac{X \overset{as\,0}{\rightsquigarrow} Y_0 \qquad X \overset{as\,1}{\rightsquigarrow} Y_1}{X \mid !X \overset{b}{\rightsquigarrow} Y_0 \mid (!X \mid Y_1)}\ (\mathsf{DRL}_4)\ \left[\exists c.\ \begin{array}{l}as\,1 = c \wedge \\ \mathsf{sync}\,(as\,0)\,c\,b\end{array}\right]$$

$$\frac{X \overset{as\,0}{\rightsquigarrow} Y}{!X \overset{b}{\rightsquigarrow} !X \mid Y}\ (\mathsf{DRL}_5)\ [as\,0 = b] \qquad \frac{X \overset{as\,0}{\rightsquigarrow} Y_0 \qquad X \overset{as\,1}{\rightsquigarrow} Y_1}{!X \overset{b}{\rightsquigarrow} !X \mid (Y_0 \mid Y_1)}\ (\mathsf{DRL}_6)\ [\mathsf{sync}\,(as\,0)\,(as\,1)\,b]$$

Remarks. (1) Because the term $!X$ has only depth 1, the matched derived rules $\mathsf{DRL}_5, \mathsf{DRL}_6$ are essentially the primitive rules $\mathsf{REPL}, \mathsf{REPLS}$. Moreover, DRL_1 was obtained by a single (backwards) application of the rule PARL.

(2) Each of $\mathsf{DRL}_2, \mathsf{DRL}_3, \mathsf{DRL}_4$ arises from the composition of two primitive rules. For example, DRL_3 is obtained by applying PARR, and then applying REPLS to the resulted hypothesis:

$$\dfrac{\dfrac{X \overset{as\,0}{\rightsquigarrow} Y_0 \qquad X \overset{as\,1}{\rightsquigarrow} Y_1}{!X \overset{c}{\rightsquigarrow} !X \mid (Y_0 \mid Y_1)}\ (\mathsf{REPLS})\ [\mathsf{sync}\,(as\,0)\,(as\,1)\,c]}{X \mid !X \overset{b}{\rightsquigarrow} X \mid (!X \mid (Y_0 \mid Y_1))}\ (\mathsf{PARR})\ [c = b]$$

The side-condition of DRL_3 is obtained by composing (essentially as relations) the two side-conditions, of PARR and REPLS, yielding existential quantification over c. Of course, the side-conditions of $\mathsf{DRL}_2, \mathsf{DRL}_3, \mathsf{DRL}_4$ can be readily simplified to the equivalent forms $as\ 0 = b$, $\mathsf{sync}\,(as\,0)\,(as\,1)\,b$ and again $\mathsf{sync}\,(as\,0)\,(as\,1)\,b$, but eliminating the existential quantifiers may not be possible in general – recall that side-conditions are *arbitrary* predicates.

The only property we care about concerning elements drl of $\mathsf{mdr}_V\,U$ w.r.t. V is that $\mathsf{theYs}(drl)$ are all distinct from the variables of V. On the other hand, concerning the relationship between $\mathsf{mdr}_V\,U$ and U, we have the crucial facts of *soundness* and *completeness* w.r.t. transition:

- For all $drl \in \mathsf{mdr}_V\,U$, drl is *sound*, i.e.: for all $\tau : \mathbf{var} \to \mathbf{term}$, $as : I\!N \to \mathbf{act}$, and $b \in \mathbf{act}$, if $\tau((\mathsf{theXXs}\ drl)!j) \overset{as\,j}{\rightsquigarrow} \tau((\mathsf{theYs}\ drl)!j)$ for all $j < \mathsf{length}(\mathsf{theYs}\ drl)$ and $\mathsf{side}\ drl\ as\ b$ holds, then $(\mathsf{theS}\ drl)[\tau] \overset{b}{\rightsquigarrow} (\mathsf{theT}\ drl)[\tau]$.
- $\mathsf{mdr}_V\,U$ is *complete* for inference of transitions with sources that match U, i.e.: for all $\tau : \mathbf{var} \to \mathbf{term}$, $b \in \mathbf{act}$ and $Q \in \mathbf{term}$ such that $U[\tau] \overset{b}{\rightsquigarrow} Q$, there exist $drl \in \mathsf{mdr}_V\,U$, $\tau' : \mathbf{var} \to \mathbf{term}$ and $as : I\!N \to \mathbf{act}$ such that:

— τ' coincides with τ on $\mathsf{vars}\,U$ (hence $U[\tau] = U[\tau']$);
— $\tau'((\mathsf{theXXs}\ drl)!j) \overset{as\,j}{\rightsquigarrow} \tau'((\mathsf{theYs}\ drl)!j)$ for all $j < \mathsf{length}(\mathsf{theYs}\ drl)$;
— $\mathsf{side}\ drl\ as\ b$ holds;
— $(\mathsf{theT}\ drl)[\tau'] = Q$ (and also, remember that $\mathsf{theS}\ drl = U$).

Deduction of universal bisimulation. An *equation* will be simply a pair of terms, written $U \cong V$, and we write **equation** for the set of equations. (Note that **rel** is the same as $\mathbf{P}(\mathbf{equation})$.) Our goals will consist of pairs (set of equations) – equation, where all equations shall be thought of as being *universally quantified*. We shall mostly use S, T, U, V

for terms thought of as *patterns*, and P, Q, R for terms thought of as *instances*.

Given $U, U' \in \mathbf{term}$, $G : \mathsf{mdr}_{U'}\, U \to \mathsf{mdr}_{U}\, U'$, and $g : \prod_{drl \in \mathsf{mdr}_{U'}\, U} \{0, \ldots, \mathsf{length}(\mathsf{theXXs}(G\ drl)) - 1\} \to \{0, \ldots, \mathsf{length}(\mathsf{theXXs}\ drl) - 1\}$, we define the predicate $\mathsf{simul}\ U\ U'\ G\ g$, read "$U$ is *(one-step-)simulated* by U' via G and g", to mean that, for all $drl \in \mathsf{mdr}_{U}\, U'$, the following holds: Assume drl has the form

$$\frac{XX_0 \overset{as\ 0}{\rightsquigarrow} Y_0\ ,\ \ldots\ , XX_{n-1} \overset{as\ (n-1)}{\rightsquigarrow} Y_{n-1}}{S \overset{b}{\rightsquigarrow} T}\ [\mathsf{side}\ drl\ as\ b] \qquad (*)$$

and $drl' = G\ drl$ has the form

$$\frac{XX'_0 \overset{as\ 0}{\rightsquigarrow} Y'_0\ ,\ \ldots\ , XX'_{n'-1} \overset{as\ (n'-1)}{\rightsquigarrow} Y'_{n'-1}}{S' \overset{b}{\rightsquigarrow} T'}\ [\mathsf{side}\ drl'\ as\ b] \qquad (**)$$

(and therefore $g\ drl : \{0, \ldots, n' - 1\} \to \{0, \ldots, n - 1\}$) Then:
- (1) $XX_{g\ drl\ j} = XX'_j$ (i.e., syntactically equal, as variables) for all $j < n'$.
- (2) $\forall as : I\!N \to \mathbf{act}, b \in \mathbf{act}.\ \mathsf{side}\ drl\ as\ b \implies \mathsf{side}\ (G\ drl)\ (as \circ (g\ drl))\ b$.

Given the rules drl, of the form $(*)$, and drl', of the form $(**)$, and given $h : \{0, \ldots, n' - 1\} \to \{0, \ldots, n - 1\}$, we define $\mathsf{newGoal}\ drl\ drl'\ h$ to be the equation $T \cong T'[(Y'_j/Y_{h\,j})_{j<n'}]$, where $(Y'_j/Y_{h\,j})_{j<n'}$ is a substitution that maps each variable Y'_j to the variable $Y_{h\,j}$ (more accurately, to the term $\mathsf{Var}\ Y_{h\,j}$).

simul and newGoal will work in tandem in our deduction system as follows: Given a goal $U \cong U'$, we wish to prove U and U' universally bisimilar. For this, we should show that, for any continuation of an instance of U, there exists a bisimilar continuation of an instance of U' (and vice versa, but next we ignore the "vice versa" part). By the completeness of mdr, any transition of an instance of U is given by a derived rule drl in $\mathsf{mdr}_{U'}\, U$. By the soundness of mdr, for finding a transition of an instance of U' that simulates that of U, it would suffice to find for drl a derived rule in drl' which is possible whenever drl is possible. Thus, we first need a map $G : \mathsf{mdr}_{U'}\, U \to \mathsf{mdr}_{U}\, U'$ (giving the drl' for each $drl \in \mathsf{mdr}_{U'}\, U$), and then, for each drl, a justification of the possibility of $G\ drl$ in terms of that of drl. Now, possibility of (a transition along) a derived rule is given by its (formal) hypotheses and its side conditions. Hence, a justification of the possibility of $G\ drl$ in terms of the possibility of drl can be given by a map from the hypotheses of $G\ drl$ to those of drl that *preserves the sources* (which are variables) and *yields an implication between the side conditions* – this is formally achieved by a function $g : \prod_{drl \in \mathsf{mdr}_{U'}\, U} \{0, \ldots, \mathsf{length}(\mathsf{theXXs}(G\ drl)) - 1\} \to \{0, \ldots, \mathsf{length}(\mathsf{theXXs}\ drl) - 1\}$ that, together with G, satisfies the conditions defining $\mathsf{simul}\ U\ U'\ G\ g$. Moreover, we have to prove that, for each combination $(drl, G\ drl)$, the resulted continuations of the presumptive instances of U and U' are again bisimilar – we obtain a $\mathsf{newGoal}\ drl\ (G\ drl)\ (g\ drl)$ for each such combination (note that generating this new goal has to take into consideration the

dispatching of formal hypotheses performed by $g\ drl$, meaning that we also have to substitute some "Ys"). Finally, the incremental nature of our coinduction (inherited from the previous section) shows up: for proving each of the new goals, we may *assume* the old goal, $U \cong U'$.

We are led to the deduction relation $\vdash : \mathbf{P}(\mathbf{equation}) \to \mathbf{equation} \to \mathbf{bool}$ (with infix notation), defined inductively by the following clauses:

$$\frac{\cdot}{\theta \vdash U \cong U'}\ (\mathsf{Eqnl})\ [\theta \cup \mathsf{bis} \vdash_{\mathrm{eq}} U \cong U']$$

$$\frac{\begin{array}{l}\forall drl \in \mathsf{mdr}_{U'}\, U.\ \theta \cup \{U \cong U'\} \vdash \mathsf{newGoal}\ drl\ (G\ drl)\ (g\ drl) \\ \forall drl' \in \mathsf{mdr}_{U}\, U'.\ \theta \cup \{U \cong U'\} \vdash \mathsf{newGoal}\ drl'\ (G'\ drl')\ (g'\ drl')\end{array}}{\theta \vdash U \cong U'}\ (\mathsf{Coind})\ \left[\begin{array}{l}\mathsf{simul}\ U\ U'\ G\ g \\ \mathsf{simul}\ U'\ U\ G'\ g'\end{array}\right]$$

In the side-condition at (Eqnl), $\vdash_{\mathrm{eq}}$ is standard equational-logic deduction. We include bis among the hypotheses, because we wish to allow any known facts about bisimilarity to "help" $\vdash$-deduction, including facts obtained by means other than $\vdash$. Again, due to circularity (moving goals to the hypotheses), a rule like (Coind) cannot be sound in itself, but again we have global soundness:

Theorem 4.5 *If $\emptyset \vdash U \cong U'$, then $(U, U') \in \mathsf{ubis}$.*

Proof sketch. We use the soundness of $\vdash\!\!\!-$ (Theorem 4.1) together with the rules defining $\vdash$ being simulated by those defining $\vdash\!\!\!-$. Namely, we show, by induction on $\vdash$, that $\theta \vdash U \cong U'$ implies $\mathsf{sstvsmCl}(\theta) \vdash\!\!\!- \mathsf{sstvsmCl}(\{(U, U')\})$, where $\mathsf{sstvsmCl} : \mathbf{rel} \to \mathbf{rel}$ gives the *substitutive and symmetric closure of a relation*, i.e., $\mathsf{sstvsmCl}(\xi) = \{(S[\sigma], T[\sigma]).\ \sigma : \mathbf{var} \to \mathbf{term},\ (S, T) \in \xi\ \vee\ (T, S) \in \xi\}$.

If $\theta \vdash U \cong U'$ followed by an application of (Eqnl), then $\mathsf{sstvsmCl}(\theta) \vdash\!\!\!- \mathsf{sstvsmCl}(\{(U, U')\})$ follows applying the $\vdash\!\!\!-$-clause (Ax), since the equational closure coincides with the substitutive symmetric closure of the congruence closure.

Assume now $\theta \vdash U \cong U'$ followed by (Coind), meaning that there exist G, g, G', g' such that: **(i)** $\mathsf{simul}\ U\ U'\ G\ g$; **(ii)** $\forall drl \in \mathsf{mdr}_{U'}\, U.\ \theta \cup \{U \cong U'\} \vdash \mathsf{newGoal}\ drl\ (G\ drl)\ (g\ drl)$; **(iii)** $\mathsf{simul}\ U'\ U\ G'\ g'$; **(iv)** $\forall drl' \in \mathsf{mdr}_{U}\, U'.\ \theta \cup \{U \cong U'\} \vdash \mathsf{newGoal}\ drl'\ (G'\ drl')\ (g'\ drl')$. Then, by the induction hypothesis:

- $\forall drl \in \mathsf{mdr}_{U'}\, U.\ \mathsf{sstvsmCl}(\theta \cup \{U \cong U'\}) \vdash\!\!\!- \mathsf{sstvsmCl}(\{\mathsf{newGoal}\ drl\ (G\ drl)\ (g\ drl)\})$.
- $\forall drl' \in \mathsf{mdr}_{U}\, U'.\ \mathsf{sstvsmCl}(\theta \cup \{U \cong U'\}) \vdash\!\!\!- \mathsf{sstvsmCl}(\mathsf{newGoal}\ drl'\ (G'\ drl')\ (g'\ drl'))$.

Let $\theta' = \mathsf{sstvsmCl}(\{(U, U')\})$ and let $\theta'' = \{\mathsf{newGoal}\ drl\ (G\ drl)\ (g\ drl).\ drl \in \mathsf{mdr}_{U'}\, U\} \cup \{\mathsf{newGoal}\ drl'\ (G'\ drl')\ (g'\ drl').\ drl' \in \mathsf{mdr}_{U}\, U'\}$. The crucial thing to notice is that, since $\mathsf{simul}\ U\ U'\ G\ g$ and $\mathsf{simul}\ U'\ U\ G'\ g'$ hold, $\mathsf{sstvsmCl}(\{(U, U')\}) \subseteq \mathsf{Retr}\ \theta'$ also holds – and the paragraph right before introducing $\vdash$ can be regarded as an informal justification for why this is true. Therefore, θ'' is an "interpolant" for applying the $\vdash\!\!\!-$-clause (Coind). Indeed, applying the $\vdash\!\!\!-$-clause (Split) to (1) and (2), we obtain $\theta' \cup \theta \vdash\!\!\!- \theta''$ and then, by the $\vdash\!\!\!-$-clause (Coind), we obtain $\theta \vdash\!\!\!- \theta'$, as desired. $\square$

Running example (finished). We are now ready to make rigorous the proof of $\forall P \in$ **term**. $(P|!P, !P) \in \mathsf{bis}$ presented in the introduction. Consider the following four proof trees of depth 0 (later referred to as Pf_1, Pf_2, Pf_3, Pf_4) where we list the side-conditions for (Eqnl) as hypotheses:

$$\frac{\{X|!X \cong !X\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} Y|!X \cong !X|Y}{\{X|!X \cong !X\} \vdash Y|!X \cong !X|Y}(\mathsf{Eqnl})$$

$$\frac{\{X|!X \cong !X\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} X|(!X|Y) \cong !X|Y}{\{X|!X \cong !X\} \vdash X|(!X|Y) \cong !X|Y}(\mathsf{Eqnl})$$

$$\frac{\{X|!X \cong !X\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} X|(!X|(Y_0|Y_1)) \cong !X|(Y_0|Y_1)}{\{X|!X \cong !X\} \vdash X|(!X|(Y_0|Y_1)) \cong !X|(Y_0|Y_1)}(\mathsf{Eqnl})$$

$$\frac{\{X|!X \cong !X\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} Y_0|(!X|Y_1) \cong !X|(Y_0|Y_1)}{\{X|!X \cong !X\} \vdash Y_0|(!X|Y_1) \cong !X|(Y_0|Y_1)}(\mathsf{Eqnl})$$

Then our final proof (tree) is:

$$\frac{Pf_1 \quad Pf_2 \quad Pf_3 \quad Pf_4}{\emptyset \vdash X|!X \cong !X}(\mathsf{Coind})$$

Explanations. At (Coind), we took:

- G to map DRL_1 and DRL_2 to DRL_5, and to map DRL_3 and DRL_4 to DRL_6;
- g DRL_1 and g DRL_2 to be the identity on $\{0\}$, and g DRL_3 and g DRL_4 to be the identity on $\{0, 1\}$;
- G' to map DRL_5 to DRL_1, and to map DRL_6 to DRL_3;
- g' DRL_5 to be the identity on $\{0\}$, and g' DRL_6 to be the identity on $\{0, 1\}$.

(Note that any function G' mapping DRL_5 to either DRL_1 or DRL_2 and DRL_6 to either DRL_3 or DRL_4 together with g' as above would lead to a valid proof.)

Here is why we end up with the above four proof tasks after applying (Coind):

$\mathsf{newGoal}\,\mathsf{DRL}_1(G\,\mathsf{DRL}_1)(g\,\mathsf{DRL}_1) = \mathsf{newGoal}\,\mathsf{DRL}_1\,\mathsf{DRL}_5(\lambda i.\, i) = Y|!X \cong !X|Y$;

$\mathsf{newGoal}\,\mathsf{DRL}_2(G\,\mathsf{DRL}_2)(g\,\mathsf{DRL}_2) = \mathsf{newGoal}\,\mathsf{DRL}_2\,\mathsf{DRL}_5(\lambda i.\, i) = X|(!X|Y) \cong !X|Y$;

$\mathsf{newGoal}\,\mathsf{DRL}_3(G\,\mathsf{DRL}_3)(g\,\mathsf{DRL}_3) = \mathsf{newGoal}\,\mathsf{DRL}_3\mathsf{DRL}_6(\lambda i.\, i) = X|(!X|(Y_0|Y_1)) \cong !X|(Y_0|Y_1)$;

$\mathsf{newGoal}\,\mathsf{DRL}_4(G\,\mathsf{DRL}_4)(g\,\mathsf{DRL}_4) = \mathsf{newGoal}\,\mathsf{DRL}_4\,\mathsf{DRL}_6(\lambda i.\, i) = Y_0|(!X|Y_1) \cong !X|(Y_0|Y_1)$;

$\mathsf{newGoal}\,\mathsf{DRL}_5(G'\mathsf{DRL}_5)(g'\mathsf{DRL}_5) = \mathsf{newGoal}\,\mathsf{DRL}_5\,\mathsf{DRL}_1(\lambda i.\, i) = Y|!X \cong !X|Y$;

$\mathsf{newGoal}\,\mathsf{DRL}_6(G'\mathsf{DRL}_6)(g'\mathsf{DRL}_6) = \mathsf{newGoal}\,\mathsf{DRL}_6\mathsf{DRL}_3(\lambda i.\, i) = X|(!X|(Y_0|Y_1)) \cong !X|(Y_0|Y_1)$.

The side-conditions of (Coind) are immediately checkable. E.g., for $\mathsf{simul}\ (X|!X)\ (!X)\ G\ g$, we need to check the following trivial facts:

- w.r.t. condition (1) (in the definition of simul): that $X = X$.
- w.r.t. condition (2): that each of the following are pairwise equivalent:

— $as\ 0 = b$ and $as\ 0 = b$;

— $\exists c.\ as\ 0 = c \wedge c = b$ and $as\ 0 = b$;

— $\exists c.\ \mathsf{sync}\,(as\ 0)\,(as\ 1)\,c \wedge c = b$ and $\mathsf{sync}\,(as\ 0)\,(as\ 1)\,b$;
— $\exists c.\ as\ 1 = c \wedge \mathsf{sync}\,(as\ 0)\,c\,b$ and $\mathsf{sync}\,(as\ 0)\,(as\ 1)\,b$.

At (Eqnl) in all the four immediate subtrees of the main proof tree, we considered the fact (assumed previously proved) that $\{X|Y \cong Y|X,\ (X|Y)|Z \cong X|(Y|Z)\} \subseteq \mathsf{bis}$, hence what we really used was equational-logic deduction from $\{X|!X \cong\ !X,\ X|Y \cong Y|X,\ (X|Y)|Z \cong X|(Y|Z)\}$, which easily discharges the equational side-conditions of the axioms, finalizing the proof.

The above proof does not display any non-trivial "dispatch" function g in the (Coind) rule application. In general however, it is not guaranteed that the formal hypotheses of two obtained derived rules (from the two terms of the goal) that one wishes to pair come in the same order, nor that these rules have the same number of hypotheses. (See the proof of commutativity of "|" from Section 4.6.)

Moreover, one may wonder why do we employ in the (Coind) rule for goal $U \cong U'$ the non-symmetric technique of showing how to simulate rule-wise first U by U', and then U' by U, while in this example it seems that a rule-wise "bi-simulation" relation between the two sets of matched derived rules, namely one consisting of the pairs $\{(\mathsf{DRL}_1, \mathsf{DRL}_5),\ (\mathsf{DRL}_2, \mathsf{DRL}_5),\ (\mathsf{DRL}_3, \mathsf{DRL}_6),\ (\mathsf{DRL}_4, \mathsf{DRL}_6)\}$ is equally successful to, while more compact than, the two maps G and G'. The answer to this is that employing two maps is strictly more general than employing a relation, since in the former case one is not bound to prove that, for two chose rules, the conditions guaranteeing their possibilities are *equivalent*. However, we have not found yet interesting cases where this would count. Of course, from our "two-map" rule one can derive the simpler "one-relation" rule.

4.5 The scope of our results

As mentioned, our technique is applicable to process algebras in the de Simone format. While this format is fairly general, it is traditionally considered to exclude important cases, such as recursion and weak bisimilarity. Of course, one can ask whether our results can be extended to cope with more general formats, covering the latter cases too. While this is a legitimate question, here we explore a different one, starting from the assumption that de Simone is really the ideal format for exploring coniduction arguments incrementally: can we instead cast the above cases to (a perhaps minor variation of) de Simone? The answer is partially positive, as we briefly argue below.

4.5.1 Adding guarded recursion

We consider the following syntax for process terms, extending with a recursion operator the one given at the beginning of Section 4.2:

$$P \;::=\; \mathsf{Var}\ X \mid \mathsf{Op}\ f\ ps\ Ps \mid \mathsf{rec}\ X.\ P$$

Just like in Section 4.2, we fix *Rls*, a set of de Simone rules (not involving rec), together with the following standard collection of rules for guarded recursion:

$$\frac{P[(\mathsf{rec}\ X.\ P)/X] \overset{a}{\rightsquigarrow} P'}{\mathsf{rec}\ X.\ P \overset{a}{\rightsquigarrow} P'}\ (\mathsf{Rec}) \quad [X \text{ guarded in } P]$$

where the guardedness assumption means: any occurrence of X in P is under an operator f which is an *Rls*-guard, the latter meaning that the rules $rl \in Rls$ with $\mathsf{thef}\ rl = f$ (i.e., having f on the left of their conclusion) are axioms (i.e., have an empty list of hypotheses). This is a generalization of CCS guardedness [87], taken from [16] – according to this general definition, the only guards in CCS are 0 (which is useless, since, being a constant, cannot "guard" anything) and the prefix operators. [16] also shows that (Rec) can be replaced (without affecting derivability) by the following more amenable rule, which takes advantage of guardedness:

$$\frac{P \overset{a}{\rightsquigarrow} P'}{\mathsf{rec}\ X.\ P \overset{a}{\rightsquigarrow} P'[(\mathsf{rec}\ X.\ P)/X]}\ (\mathsf{Rec}_2) \quad [X \text{ guarded in } P]$$

(Intuitively, this is because, thanks to guardedness, the term $P[(\mathsf{rec}\ X.\ P)/X]$ from the hypothesis of (Rec) will not involve the "$\mathsf{rec}\ X.\ P$" part in any (one-step) transition, and therefore substituting $\mathsf{rec}\ X.\ P$ for X *before* the transition, as in (Rec), has the same effect as substituting it *after* the transition, as in (Rec_2) – see op. cit.)

(Rec_2) is in a de-Simone-like format, since it does express the behavior of $\mathsf{rec}\ X.\ P$ in terms of that of its (strictly smaller) component P, and our results from Sections 4.3 and 4.4 could be easily extended to cope with $Rls \cup (\mathrm{Rec}_2)$. In fact, it falls under the obvious generalization of de Simone for syntax with bindings. (And it appears that all our results from this chapter can be easily generalized to cope with this.)

4.5.2 Considering weak bisimilarity

Here we do not attempt a general answer as we did for guarded recursion. Instead, following [123], we sketch on a particular case an approach seemingly generalizable to a more abstract setting.

Let us consider our running example, the mini-CCS described in the introduction and defined more rigorously in Section 4.2. Thus, the syntax is:

$$P \ ::= \ 0 \mid a.P \mid P|Q \mid !\,P$$

and the rules are (PREF), (PARL), (PARR), (PARS), (REPL) and (REPLS).

Recall that a *weak simulation* is a relation $\theta \subseteq \mathbf{term} \times \mathbf{term}$ such that the following hold for all $(P, Q) \in \theta$:
- if $P \stackrel{\tau}{\rightsquigarrow} P'$ for some P', then there exists Q' such that $Q \stackrel{\tau^*}{\rightsquigarrow} Q'$ and $(P', Q') \in \theta$;
- if $P \stackrel{a}{\rightsquigarrow} P'$ for some P' and $a \in \mathbf{act}$, then there exists Q' such that $Q \stackrel{\tau^* a \tau^*}{\rightsquigarrow} Q'$ and $(P', Q') \in \theta$.

(where τ^* means 0 or more τ-transitions, and therefore $\tau^* a \tau^*$ means: 0 or more τ-transitions, followed by an a-transition, followed by 0 or more τ-transitions). A *weak bisimulation* is a simulation such that its converse is also a simulation, and *weak bisimilarity* is the largest weak bisimulation. Note that the usual bisimilarity relation bis (a.k.a. "strong bisimilarity"), is included in weak bisimilarity.

We wish to express weak bisimilarity as (strong) bisimilarity in another system in de Simone format (over the same syntax). For this, the usual technique of adding τ-rules for reflexivity and transitivity does not work, as the lookahead introduced by transitivity cuts all the bridges with the de Simone format. Instead, we employ the following chain of thought from [123], relevant for both a presumptive denotational semantics and a presumptive *compositional* operational semantics for weak bisimilarity (the latter being essentially our goal):
- to capture weak bisimilarity, we need to keep track of the (process) continuations along arbitrarily long traces of τ-actions;
- to keep track of the latter (in a manner compositional w.r.t. operators such as parallel composition), we need to also keep track of continuations along arbitrary traces of actions (of any kind).

This leads to a trace-based version of the system. In what follows, a ranges over loud actions (i.e., actions different from τ), w ranges over lists of loud actions, $\#$ denotes list concatenation and ϵ the empty list. Knowing how *single actions* interact, and following the interleaving semantics from the rules for the $|$ operator, we obtain the following recursive definition of the *parallel composition* (or *synchronized shuffle*) of action traces, $| : \mathbf{List}(\mathbf{act}) \times \mathbf{List}(\mathbf{act}) \to \mathbf{P}(\mathbf{List}(\mathbf{act}))$:

- $\epsilon|\epsilon = \{\epsilon\}$;
- $(a\#w_1)|(b\#w_2) = a\#(w_1|(b\#w_2)) \cup b\#((a\#w_1)|w_2)$, if $\overline{a} \neq b$;
- $(a\#w_1)|(b\#w_2) = a\#(w_1|(b\#w_2)) \cup b\#((a\#w_1)|w_2) \cup w_1|w_2$, if $\overline{a} = b$.

(Above, we overloaded $\#$ in the usual fashion, to denote both $\# : \mathbf{List}(\mathbf{act}) \times \mathbf{List}(\mathbf{act}) \to \mathbf{List}(\mathbf{act})$ and its componentwise extension to $\mathbf{List}(\mathbf{act}) \times \mathbf{P}(\mathbf{List}(\mathbf{act})) \to \mathbf{P}(\mathbf{List}(\mathbf{act}))$, given by $w \# S = \{w \# w'.\, w' \in S\}$.)

Now, we define the following transition system, whose labels are sequences of actions:

$$\frac{\cdot}{P \overset{\epsilon}{\rightsquigarrow} P}(\mathsf{Silent}) \qquad \frac{P \overset{w}{\rightsquigarrow} P'}{a.P \overset{a\#w}{\rightsquigarrow} P'}(\mathsf{PrefT})$$

$$\frac{P \overset{w_1}{\rightsquigarrow} P' \qquad Q \overset{w_2}{\rightsquigarrow} Q'}{P|Q \overset{w}{\rightsquigarrow} P'|Q'}\ (\mathsf{ParT})\ [w \in w_1|w_2] \qquad \frac{P \overset{w_1}{\rightsquigarrow} Q'_1 \quad \ldots \quad P \overset{w_n}{\rightsquigarrow} Q'_n}{!P \overset{w}{\rightsquigarrow} (!P)|Q'_1|\ldots|Q'_n}\ (\mathsf{ReplT})\ [w \in w_1|\ldots|w_n]$$

The above rules were produced by taking the reflexive-transitive closure of the rules of the original system, i.e., by composing that system with itself *horizontally* an indefinite number of times. In the process, we also made sure that zero or more τ actions were identified with the empty trace ϵ, and in particular we have added the rule (Silent).

One can then show that two processes are weakly bisimilar in the original system iff they are (strongly) bisimilar in the trace-based system – see [123]. Moreover, the latter system has only de Simone rules, plus the rule (Silent), which is reducible to the de Simone format by simply traversing the terms recursively for each term construct:

$$\frac{\cdot}{0 \overset{\epsilon}{\rightsquigarrow} 0}(\mathsf{Silent}_0) \qquad \frac{P \overset{\epsilon}{\rightsquigarrow} P'}{a.P \overset{\epsilon}{\rightsquigarrow} a.P'}(\mathsf{Silent}_a)$$

etc.

4.5.3 Combining guarded recursion with weak bisimilarity?

Note that the amenable example from Section 4.5.2 already contains a restrictive form of (π-calculus like) recursion. Unfortunately however, general guarded recursion does not interact well with weak bisimilarity, in that guardedness is ineffective when observing arbitrarily long traces.

Thus, although the recursion rule (Rec) (even without assuming guardedness) can be soundly made, by the technique from Section 4.5.2, into an (identically-shaped) trace-based rule

$$\frac{P[(\mu X.\, P)/X] \overset{w}{\rightsquigarrow} P'}{\mu X.\, P \overset{w}{\rightsquigarrow} P'}(\mathsf{RecT})$$

the technique for reducing recursion to guarded recursion from Section 4.5.1 (crucial for converting to de Simone) does not work, or, more precisely, is vacuous for the trace-based system. Indeed, in the latter system, the rule for prefix no longer being an axiom, the prefix operators are no longer guards.

4.6 More examples

This section presents more examples (in full technical detail) illustrating the use of our coinductive proof system.

4.6.1 The proofs of commutativity and associativity of | in the mini process calculus

Here we work in the context of the running example from this chapter.

Commutativity. The proof of $\forall P, Q \in \mathbf{term}.\ (P|Q, Q|P) \in bis$, i.e., of $\emptyset \vdash X_0|X_1 \cong X_1|X_0$, goes as follows (where again we list the side-conditions for (Eqnl) as hypotheses, and where the three (Eqnl)-rooted proof trees are subtrees of the main, (Coind)-rooted proof tree):

$$\dfrac{\dfrac{\dfrac{}{\{X_0|X_1 \cong X_1|X_0\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} Y_0|X_1 \cong X_1|Y_0}(\mathsf{Eqnl})}{\{X_0|X_1 \cong X_1|X_0\} \vdash Y_0|X_1 \cong X_1|Y_0}(\mathsf{Eqnl})}{\emptyset \vdash X_0|X_1 \cong X_1|X_0}(\mathsf{Coind})$$

$$\dfrac{\dfrac{\dfrac{}{\{X_0|X_1 \cong X_1|X_0\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} X_0|Y_1 \cong Y_1|X_0}(\mathsf{Eqnl})}{\{X_0|X_1 \cong X_1|X_0\} \vdash X_0|Y_1 \cong Y_1|X_0}(\mathsf{Eqnl})}{\emptyset \vdash X_0|X_1 \cong X_1|X_0}(\mathsf{Coind})$$

$$\dfrac{\dfrac{\dfrac{}{\{X_0|X_1 \cong X_1|X_0\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} Y_0|Y_1 \cong Y_1|Y_0}(\mathsf{Eqnl})}{\{X_0|X_1 \cong X_1|X_0\} \vdash Y_0|Y_1 \cong Y_1|Y_0}(\mathsf{Eqnl})}{\emptyset \vdash X_0|X_1 \cong X_1|X_0}(\mathsf{Coind})$$

Explanations: Each of the (Eqnl) rules discharges the goal immediately by (trivial) equational-logic reasoning.

We have that:

- $\mathsf{mdr}_{X_1|X_0}(X_0|X_1) = \{\mathsf{DRL}_1, \mathsf{DRL}_2, \mathsf{DRL}_3\}$ and
- $\mathsf{mdr}_{X_0|X_1}(X_1|X_0) = \{\mathsf{DRL}_4, \mathsf{DRL}_5, \mathsf{DRL}_6\}$, where:

$$\dfrac{X_0 \overset{as\,0}{\rightsquigarrow} Y_0}{X_0 \mid X_1 \overset{b}{\rightsquigarrow} Y_0 \mid X_1}\ (\mathsf{DRL}_1)\ [as\,0 = b] \qquad \dfrac{X_1 \overset{as\,0}{\rightsquigarrow} Y_1}{X_0 \mid X_1 \overset{b}{\rightsquigarrow} X_0 \mid Y_1}\ (\mathsf{DRL}_2)\ [as\,0 = b] \qquad \dfrac{X_0 \overset{as\,0}{\rightsquigarrow} Y_0 \quad X_1 \overset{as\,1}{\rightsquigarrow} Y_1}{X_0 \mid X_1 \overset{b}{\rightsquigarrow} Y_0 \mid Y_1}\ (\mathsf{DRL}_3)\ [\mathsf{sync}\,(as\,0)\,(as\,1)\,b]$$

$$\dfrac{X_1 \overset{as\,0}{\rightsquigarrow} Y_1}{X_1 \mid X_0 \overset{b}{\rightsquigarrow} Y_1 \mid X_0}\ (\mathsf{DRL}_4)\ [as\,0 = b] \qquad \dfrac{X_0 \overset{as\,0}{\rightsquigarrow} Y_0}{X_1 \mid X_0 \overset{b}{\rightsquigarrow} X_1 \mid Y_0}\ (\mathsf{DRL}_5)\ [as\,0 = b] \qquad \dfrac{X_1 \overset{as\,0}{\rightsquigarrow} Y_1 \quad X_0 \overset{as\,1}{\rightsquigarrow} Y_0}{X_1 \mid X_0 \overset{b}{\rightsquigarrow} Y_1 \mid Y_0}\ (\mathsf{DRL}_6)\ [\mathsf{sync}\,(as\,0)\,(as\,1)\,b]$$

(Since the terms $X|Y$ and $Y|X$ consist of an operation applied to distinct variables, the matched derived rules of either of them are, up to a renaming, (PARL), (PARR) and (PARS). But of course this does not mean that the terms are a priori bisimilar, since the renamings matter. E.g., if the mini calculus lacked (PARR), the two terms would not be bisimilar.)

At (Coind), we took:

- G to map DRL_1 to DRL_5, DRL_2 to DRL_4, and DRL_3 to DRL_6;
- g DRL_1 and g DRL_2 to be the identity map on $\{0\}$, and g DRL_3 to be $\lambda i : \{0,1\}.\ 1 - i$;
- G' to map DRL_4 to DRL_2, DRL_5 to DRL_1, and DRL_6 to DRL_3;

- g' DRL_4 and g' DRL_5 to be the identity map on $\{0\}$, and g DRL_6 to be $\lambda i \in \{0,1\}.\, 1-i$.

(Note that g DRL_3 and g DRL_6 are cases of nontrivial "dispatch" maps.)

Here is why we end up with the above three proof tasks after applying (Coind) backwards:

- $\mathsf{newGoal}\,\mathsf{DRL}_1\,(G\,\mathsf{DRL}_1)\,(g\,\mathsf{DRL}_1) = \mathsf{newGoal}\,\mathsf{DRL}_1\,\mathsf{DRL}_5\,(\lambda i \in \{0\}.\, i) = Y_0|X_1 \cong X_1|Y_0$;
- $\mathsf{newGoal}\,\mathsf{DRL}_2\,(G\,\mathsf{DRL}_2)\,(g\,\mathsf{DRL}_2) = \mathsf{newGoal}\,\mathsf{DRL}_2\,\mathsf{DRL}_4\,(\lambda i \in \{0\}.\, i) = X_0|Y_1 \cong Y_1|X_0$;
- $\mathsf{newGoal}\,\mathsf{DRL}_3\,(G\,\mathsf{DRL}_3)\,(g\,\mathsf{DRL}_3) = \mathsf{newGoal}\,\mathsf{DRL}_3\,\mathsf{DRL}_6\,(\lambda i \in \{0,1\}.\, 1-i) = Y_0|Y_1 \cong Y_1|Y_0$;
- $\mathsf{newGoal}\,\mathsf{DRL}_4\,(G'\,\mathsf{DRL}_4)\,(g'\,\mathsf{DRL}_4) = \mathsf{newGoal}\,\mathsf{DRL}_4\,\mathsf{DRL}_2\,(\lambda i \in \{0\}.\, i) = X_0|Y_1 \cong Y_1|X_0$;
- $\mathsf{newGoal}\,\mathsf{DRL}_5\,(G'\,\mathsf{DRL}_5)\,(g'\,\mathsf{DRL}_5) = \mathsf{newGoal}\,\mathsf{DRL}_5\,\mathsf{DRL}_1\,(\lambda i \in \{0\}.\, i) = Y_0|X_1 \cong X_1|Y_0$;
- $\mathsf{newGoal}\,\mathsf{DRL}_6\,(G'\,\mathsf{DRL}_6)\,(g'\,\mathsf{DRL}_6) = \mathsf{newGoal}\,\mathsf{DRL}_6\,\mathsf{DRL}_3\,(\lambda i \in \{0,1\}.\, 1-i) = Y_0|Y_1 \cong Y_1|Y_0$.

The side-conditions of (Coind) are verified as follows:

- $\mathsf{simul}\ (X_0|X_1)\ (X_1|X_0)\ G\ g$ amounts to the following:

— $as\,0 = b \iff as\,0 = b$, trivial;

— $\mathsf{sync}\,(as\ 0)\,(as\ 1)\,b \iff \mathsf{sync}\,(as\ (g\ \mathsf{DRL}_3\ 0))\,(as\ (g\ \mathsf{DRL}_3\ 1))\,b$, i.e.,

$\mathsf{sync}\,(as\ 0)\,(as\ 1)\,b \iff \mathsf{sync}\,(as\ 1)\,(as\ 0)\,b$, which follows by the definition of sync and the assumption that $\forall a \in \mathbf{act}.\, \overline{\overline{a}} = a$.

- similarly, $\mathsf{simul}\ (X_1|X_0)\ (X_1|X_0)\ G\ g$ amounts to the following:

— $as\,0 = b \iff as\,0 = b$, trivial;

— $\mathsf{sync}\,(as\ 0)\,(as\ 1)\,b \iff \mathsf{sync}\,(as\ (g'\ \mathsf{DRL}_6\ 0))\,(as\ (g'\ \mathsf{DRL}_6\ 1))\,b$, i.e., again,

$\mathsf{sync}\,(as\ 0)\,(as\ 1)\,b \iff \mathsf{sync}\,(as\ 1)\,(as\ 0)\,b$.

Associativity. The proof of $\forall P, Q, R \in \mathbf{term}.\ ((P|Q)|R, P|(Q|R)) \in bis$, i.e., of $\emptyset \vdash (X_0|X_1)|X_2 \cong X_0|(X_1|X_2))$, goes as follows (where again we list the side-conditions for (Eqnl) as hypotheses, and where the six (Eqnl)-rooted proof trees are subtrees of the main, (Coind)-rooted proof tree):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} (Y_0|X_1)|X_2 \cong Y_0|(X_1|X_2)}(\mathsf{Eqnl})}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \vdash (Y_0|X_1)|X_2 \cong Y_0|(X_1|X_2)}(\mathsf{Eqnl})}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} (X_0|Y_1)|X_2 \cong X_0|(Y_1|X_2)}(\mathsf{Eqnl})}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \vdash (X_0|Y_1)|X_2 \cong X_0|(Y_1|X_2)}(\mathsf{Eqnl})}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} (Y_0|Y_1)|X_2 \cong Y_0|(Y_1|X_2)}(\mathsf{Eqnl})}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \vdash (Y_0|Y_1)|X_2 \cong Y_0|(Y_1|X_2)}(\mathsf{Eqnl})}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} (Y_0|X_1)|Y_2 \cong Y_0|(X_1|Y_2)}(\mathsf{Eqnl})}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \vdash Y|!X \cong (Y_0|X_1)|Y_2 \cong Y_0|(X_1|Y_2)}(\mathsf{Eqnl})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} (X_0|Y_1)|Y_2 \cong X_0|(Y_1|Y_2)}(\mathsf{Eqnl})}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \vdash (X_0|Y_1)|Y_2 \cong X_0|(Y_1|Y_2)}(\mathsf{Eqnl})}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} (X_0|X_1)|Y_2 \cong X_0|(X_1|Y_2)}(\mathsf{Eqnl})}{\{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)\} \vdash (X_0|X_1)|Y_2 \cong X_0|(X_1|Y_2)}(\mathsf{Eqnl})}{(X_0|X_1)|X_2 \cong X_0|(X_1|X_2)}(\mathsf{Coind})$$

Explanations: Each of the (Eqnl) rules discharges the goal immediately by (trivial) equational-

logic reasoning.

We have that:

- $\mathsf{mdr}_{X_0|(X_1|X_2)}((X_0|X_1)|X_2) = \{\mathsf{DRL}_1, \mathsf{DRL}_2, \mathsf{DRL}_3, \mathsf{DRL}_4, \mathsf{DRL}_5, \mathsf{DRL}_6\}$,

where:

$$\frac{X_0 \overset{as\ 0}{\rightsquigarrow} Y_0}{(X_0|X_1)|X_2 \overset{b}{\rightsquigarrow} (Y_0|X_1)|X_2}\ (\mathsf{DRL}_1)\ \left[\exists c.\ \begin{array}{l} as\ 0 = c \\ \wedge\ c = b \end{array}\right] \qquad \frac{X_1 \overset{as\ 0}{\rightsquigarrow} Y_1}{(X_0|X_1)|X_2 \overset{b}{\rightsquigarrow} (X_0|Y_1)|X_2}\ (\mathsf{DRL}_2)\ \left[\exists c.\ \begin{array}{l} as\ 0 = c \\ \wedge\ c = b \end{array}\right]$$

$$\frac{X_0 \overset{as\ 0}{\rightsquigarrow} Y_0 \quad X_1 \overset{as\ 1}{\rightsquigarrow} Y_1}{(X_0|X_1)|X_2 \overset{b}{\rightsquigarrow} (Y_0|Y_1)|X_2}\ (\mathsf{DRL}_3)\ \left[\exists c.\ \begin{array}{l} \mathsf{sync}\,(as\ 0)\,(as\ 1)\,c \\ \wedge\ c = b \end{array}\right] \qquad \frac{X_0 \overset{as\ 0}{\rightsquigarrow} Y_0 \quad X_2 \overset{as\ 1}{\rightsquigarrow} Y_2}{(X_0|X_1)|X_2 \overset{b}{\rightsquigarrow} (Y_0|X_1)|Y_2}\ (\mathsf{DRL}_4)\ \left[\exists c.\ \begin{array}{l} as\ 0 = c \wedge \\ \mathsf{sync}\,c\,(as\ 1)\,b \end{array}\right]$$

$$\frac{X_1 \overset{as\ 0}{\rightsquigarrow} Y_1 \quad X_2 \overset{as\ 1}{\rightsquigarrow} Y_2}{(X_0|X_1)|X_2 \overset{b}{\rightsquigarrow} (X_0|Y_1)|Y_2}\ (\mathsf{DRL}_5)\ \left[\exists c.\ \begin{array}{l} as\ 0 = c \wedge \\ \mathsf{sync}\,c\,(as\ 1)\,b \end{array}\right] \qquad \frac{X_2 \overset{as\ 0}{\rightsquigarrow} Y_2}{(X_0|X_1)|X_2 \overset{b}{\rightsquigarrow} (X_0|X_1)|Y_2}\ (\mathsf{DRL}_6)\ [as\ 0 = b]$$

- $\mathsf{mdr}_{(X_0|X_1)|X_2}(X_0|(X_1|X_2)) = \{\mathsf{DRL}_7, \mathsf{DRL}_8, \mathsf{DRL}_9, \mathsf{DRL}_{10}, \mathsf{DRL}_{11}, \mathsf{DRL}_{12}\}$, where:

$$\frac{X_0 \overset{as\ 0}{\rightsquigarrow} Y_0}{X_0|(X_1|X_2) \overset{b}{\rightsquigarrow} Y_0|(X_1|X_2)}\ (\mathsf{DRL}_7)\ [as\ 0 = b] \qquad \frac{X_1 \overset{as\ 0}{\rightsquigarrow} Y_1}{X_0|(X_1|X_2) \overset{b}{\rightsquigarrow} X_0|(Y_1|X_2)}\ (\mathsf{DRL}_8)\ \left[\exists c.\ \begin{array}{l} as\ 0 = c \\ \wedge\ c = b \end{array}\right]$$

$$\frac{X_0 \overset{as\ 0}{\rightsquigarrow} Y_0 \quad X_1 \overset{as\ 1}{\rightsquigarrow} Y_1}{X_0|(X_1|X_2) \overset{b}{\rightsquigarrow} Y_0|(Y_1|X_2)}\ (\mathsf{DRL}_9)\ \left[\exists c.\ \begin{array}{l} as\ 0 = c \wedge \\ \mathsf{sync}\,(as\ 0)\,c\,b \end{array}\right] \qquad \frac{X_0 \overset{as\ 0}{\rightsquigarrow} Y_0 \quad X_2 \overset{as\ 1}{\rightsquigarrow} Y_2}{X_0|(X_1|X_2) \overset{b}{\rightsquigarrow} Y_0|(X_1|Y_2)}\ (\mathsf{DRL}_{10})\ \left[\exists c.\ \begin{array}{l} as\ 1 = c \wedge \\ \mathsf{sync}(as\ 0)\,c\,b \end{array}\right]$$

$$\frac{X_1 \overset{as\ 0}{\rightsquigarrow} Y_1 \quad X_2 \overset{as\ 1}{\rightsquigarrow} Y_2}{X_0|(X_1|X_2) \overset{b}{\rightsquigarrow} X_0|(Y_1|Y_2)}\ (\mathsf{DRL}_{11})\ \left[\exists c.\ \begin{array}{l} \mathsf{sync}\,(as\ 0)\,(as\ 1)\,c \\ \wedge\ c = b \end{array}\right] \qquad \frac{X_2 \overset{as\ 0}{\rightsquigarrow} Y_2}{X_0|(X_1|X_2) \overset{b}{\rightsquigarrow} X_0|(X_1|Y_2)}\ (\mathsf{DRL}_{12})\ \left[\exists c.\ \begin{array}{l} as\ = c \\ \wedge\ c = b \end{array}\right]$$

At (Coind), we took:

- G to map each DRL_i, with $i \in \{1, \ldots, 6\}$, to DRL_{i+6};
- all the g DRL_i, with $i \in \{1, \ldots, 6\}$, to be identity maps;
- G' to map each DRL_j, with $j \in \{7, \ldots, 12\}$, to DRL_{j-6};
- all the g' DRL_j, with $j \in \{7, \ldots, 12\}$, to be identity maps.

Here is why we end up with the above six proof tasks after applying (Coind) backwards:

- $\mathsf{newGoal}\,\mathsf{DRL}_1\,(G\,\mathsf{DRL}_1)\,(g\,\mathsf{DRL}_1) = \mathsf{newGoal}\,\mathsf{DRL}_1\,\mathsf{DRL}_7\,(\lambda i.\,i) = (Y_0|X_1)|X_2 \cong Y_0|(X_1|X_2)$;
- $\mathsf{newGoal}\,\mathsf{DRL}_2\,(G\,\mathsf{DRL}_2)\,(g\,\mathsf{DRL}_2) = \mathsf{newGoal}\,\mathsf{DRL}_2\,\mathsf{DRL}_8\,(\lambda i.\,i) = (X_0|Y_1)|X_2 \cong X_0|(Y_1|X_2)$;
- $\mathsf{newGoal}\,\mathsf{DRL}_3\,(G\,\mathsf{DRL}_3)\,(g\,\mathsf{DRL}_3) = \mathsf{newGoal}\,\mathsf{DRL}_3\,\mathsf{DRL}_9\,(\lambda i.\,i) = (Y_0|Y_1)|X_2 \cong Y_0|(Y_1|X_2)$;
- $\mathsf{newGoal}\,\mathsf{DRL}_4\,(G'\,\mathsf{DRL}_4)\,(g'\,\mathsf{DRL}_4) = \mathsf{newGoal}\,\mathsf{DRL}_4\,\mathsf{DRL}_{10}\,(\lambda i.\,i) = (Y_0|X_1)|Y_2 \cong Y_0|(X_1|Y_2)$;
- $\mathsf{newGoal}\,\mathsf{DRL}_5\,(G'\,\mathsf{DRL}_5)\,(g'\,\mathsf{DRL}_5) = \mathsf{newGoal}\,\mathsf{DRL}_5\,\mathsf{DRL}_{11}\,(\lambda i.\,i) = (X_0|Y_1)|Y_2 \cong X_0|(Y_1|Y_2)$;
- $\mathsf{newGoal}\,\mathsf{DRL}_6\,(G'\,\mathsf{DRL}_6)\,(g'\,\mathsf{DRL}_6) = \mathsf{newGoal}\,\mathsf{DRL}_6\,\mathsf{DRL}_{12}\,(\lambda i.\,i) = (X_0|X_1)|Y_2 \cong X_0|(X_1|Y_2)$.

Moreover, we have $\mathsf{newGoal}\,\mathsf{DRL}_j\,(G'\,\mathsf{DRL}_j)\,(g'\,\mathsf{DRL}_j) = \mathsf{newGoal}\,\mathsf{DRL}_{j-6}\,(G\,\mathsf{DRL}_j)\,(g\,\mathsf{DRL}_j)$ for all $j \in \{7, \ldots, 12\}$, and therefore the above six new goals are all the new goals.

The side-conditions of (Coind), namely $\mathsf{simul}\ ((X_0|X_1)|X_2)\ (X_0|(X_1|X_2))\ G\ g$ and $\mathsf{simul}\ (X_0|(X_1|X_2))\ ((X_0|X_1)|X_2)\ G\ g$, state precisely that, for all $i \in \{1, \ldots, 6\}$, the side condition of DRL_i is equivalent with that of DRL_{i+6}, facts that are trivial to check.

4.6.2 A deterministic example

Here we show how our setting can handle deterministic situations such as the ones from [136, 77]. The case we consider is that of *formal series (i.e., infinite polynomials) of natural numbers*.

We take **act** and **param** to be $\mathbb{N}$, and **opsym** to be a three-element set, $\{\mathsf{Cons}, \mathsf{Plus}, \mathsf{Times}\}$.

We use the following abbreviations:

- X, for $\mathsf{Var}\ X$.
- $a.S$, for $\mathsf{Op}\ \mathsf{Cons}\ [a]\ S$;
- $S + T$, for $\mathsf{Op}\ \mathsf{Plus}\ []\ [S, T]$;
- $S * T$, for $\mathsf{Op}\ \mathsf{Times}\ []\ [S, T]$.

(We assume $*$ binds more strongly than $+$.)

We take *Rls* to be $\{\mathsf{CONS}\ a.\ a \in \mathbf{act}\} \cup \{\mathsf{PLUS}, \mathsf{TIMES}\}$, where:

$$\frac{\cdot}{a.X \overset{b}{\rightsquigarrow} X}\ (\mathsf{CONS}\ a)\ [a = b] \qquad \frac{X_0 \overset{a_0}{\rightsquigarrow} Y_0 \quad X_1 \overset{a_1}{\rightsquigarrow} Y_1}{X_0 + Y_0 \overset{b}{\rightsquigarrow} X_1 + Y_1}\ (\mathsf{PLUS})\ [as\ 0 + as\ 1 = b]$$

$$\frac{X_0 \overset{a_0}{\rightsquigarrow} Y_0 \quad X_1 \overset{a_1}{\rightsquigarrow} Y_1}{X_0 * Y_0 \overset{b}{\rightsquigarrow} X_0 * Y_1 + Y_0 * X1}\ (\mathsf{TIMES})\ [as\ 0 * as\ 1 = b]$$

The above system is *syntactically deterministic* in the following sense: each operation has at most one rule with the source of the conclusion containing that operation. As a consequence, for each terms U and U', $\mathsf{mdr}_{U'}(U)$ has at most one element. Hence, during $\vdash$-deduction, we have at most one choice for the functions G, g, G', g', meaning that our $\vdash$-rule (Coind) becomes entirely syntax-directed, and therefore can be applied automatically. Semantic determinism is a consequence of the syntactic determinism: $\forall P, Q, Q' \in \mathbf{term}, a \in \mathbf{act}.\ P \overset{a}{\rightsquigarrow} Q\ \wedge P \overset{a}{\rightsquigarrow} Q'\ \rightarrow Q = Q'$.

In the the following $\vdash$-proofs, we do not indicate G, g, G', g' (as they shall be the only possible ones).

(1) Commutativity of $+$. The proof of $\forall P, Q \in \mathbf{term}.\ (P + Q, Q + P) \in bis$, i.e., of $\emptyset \vdash X_0 + X_1 \cong X_1 + X_0$, goes as follows:

$$\dfrac{\dfrac{\{X_0 + X_1 \cong X_1 + X_0\} \cup \mathsf{bis} \vdash_{\mathsf{eq}} Y_0 + Y_1 \cong Y_1 + Y_0}{\{X_0 + X_1 \cong X_1 + X_0\} \vdash Y_0 + Y_1 \cong Y_1 + Y_0}(\mathsf{Eqnl})}{\emptyset \vdash X_0 + X_1 \cong X_1 + X_0}(\mathsf{Coind})$$

(Where discharging the side-conditions for (Coind) requires commutativity of natural-number addition.)

(2) Associativity of $+$. The proof of $\forall P, Q, R \in \mathbf{term}.\ ((P + Q) + R, P + (Q + R)) \in bis$, i.e., of $\emptyset \vdash (X_0 + X_1) + X_2 \cong X_0 + (X_1 + X_2)$, goes as follows:

$$\dfrac{\dfrac{\{(X_0 + X_1) + X_2 \cong X_0 + (X_1 + X_2)\} \cup \mathsf{bis} \vdash_{\mathsf{eq}} (Y_0 + Y_1) + Y_2 \cong Y_0 + (Y_1 + Y_2)}{\{(X_0 + X_1) + X_2 \cong X_0 + (X_1 + X_2)\} \vdash (Y_0 + Y_1) + Y_2 \cong Y_0 + (Y_1 + Y_2)}(\mathsf{Eqnl})}{\emptyset \vdash (X_0 + X_1) + X_2 \cong X_0 + (X_1 + X_2)}(\mathsf{Coind})$$

(Where discharging the side-conditions for (Coind) requires associativity of natural-number addition.)

(3) Commutativity of $*$. The proof of $\forall P, Q \in \mathbf{term}.\ (P * Q, Q * P) \in bis$, i.e., of $\emptyset \vdash X_0 * X_1 \cong X_1 * X_0$, goes as follows:

$$\dfrac{\dfrac{\{X_0 * X_1 \cong X_1 * X_0\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} X_0 * Y_1 + X_1 * Y_0 \cong X_1 * Y_0 + X_0 * Y_1}{\{X_0 * X_1 \cong X_1 * X_0\} \vdash X_0 * Y_1 + X_1 * Y_0 \cong X_1 * Y_0 + X_0 * Y_1}\,(\mathsf{Eqnl})}{\emptyset \vdash X_0 * X_1 \cong X_1 * X_0}\,(\mathsf{Coind})$$

(Where (Eqnl) uses that, by point (1), $(X_0 + X_1 \cong X_1 + X_0) \in \mathsf{bis}$, and where discharging the side-conditions for (Coind) requires commutativity of natural-number multiplication.)

(4) Distributivity of $*$ w.r.t $+$. The proof of $\forall P, Q, R \in \mathbf{term}.\ (P*(Q+R), P*Q+P*R) \in bis$, i.e., of $\emptyset \vdash X_0 * (X_1 + X_2) \cong X_0 * X_1 + X_0 * X_2$, goes as follows:

$$\dfrac{\dfrac{\{X_0 * (X_1 + X_2) \cong X_0 * X_1 + X_0 * X_2\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} \begin{array}{l} X_0 * (Y_1 + Y_2) + Y_0 * (X_1 + X_2) \cong \\ X_0 * Y_1 + X_1 * Y_0 + X_0 * Y_2 + X_2 * Y_0 \end{array}}{\{X_0 * (X_1 + X_2) \cong X_0 * X_1 + X_0 * X_2\} \vdash \begin{array}{l} X_0 * (Y_1 + Y_2) + Y_0 * (X_1 + X_2) \cong \\ X_0 * Y_1 + X_1 * Y_0 + X_0 * Y_2 + X_2 * Y_0 \end{array}}\,(\mathsf{Eqnl})}{\emptyset \vdash X_0 * (X_1 + X_2) \cong X_0 * X_1 + X_0 * X_2}\,(\mathsf{Coind})$$

(Where (Eqnl) uses that, by points (1) and (3), $\{X_0+X_1 \cong X_1+X_0, X_0*X_1 \cong X_1*X_0\} \subseteq \mathsf{bis}$, and where discharging the side-conditions for (Coind) requires distributivity of multiplication w.r.t. addition for natural numbers.)

(5) Associativity of $*$. The proof of $\forall P, Q, R \in \mathbf{term}.\ ((P * Q) * R, P * (Q * R)) \in bis$, i.e., of $\emptyset \vdash (X_0 * X_1) * X_2 \cong X_0 * (X_1 * X_2)$, goes as follows:

$$\dfrac{\dfrac{\{(X_0 * X_1) * X_2 \cong X_0 * (X_1 * X_2)\} \cup \mathsf{bis} \vdash_{\mathrm{eq}} \begin{array}{l} X_0 * X_1 * Y_2 + X_2 * (X_0 * Y_1 + X_1 * Y_0) \cong \\ X_0 * (X_1 * Y_2 + X_2 * Y_1) + X_1 * X_2 * Y_0 \end{array}}{\{(X_0 * X_1) * X_2 \cong X_0 * (X_1 * X_2)\} \vdash \begin{array}{l} X_0 * X_1 * Y_2 + X_2 * (X_0 * Y_1 + X_1 * Y_0) \cong \\ X_0 * (X_1 * Y_2 + X_2 * Y_1) + X_1 * X_2 * Y_0 \end{array}}\,(\mathsf{Eqnl})}{\emptyset \vdash (X_0 * X_1) * X_2 \cong X_0 * (X_1 * X_2)}\,(\mathsf{Coind})$$

(Where (Eqnl) uses that, by points (3) and (4), $\{X_0 * X_1 \cong X_1 * X_0, X_0 * (X_1 + X_2) \cong X_0 * X_1 + X_0 * X_2\} \subseteq \mathsf{bis}$, and where discharging the side-conditions for (Coind) requires associativity of natural-number multiplication.)

4.7 Details regarding the Isabelle formalization

The main results of this chapter, namely, those from Sections 4.3 and 4.4, have been formalized in Isabelle/HOL. The formal scripts can be found at [122], where:

- document.pdf contains all theories with full formal proofs of the theorems,
- outline.pdf contains the theories with the proofs omitted,

- index.html is the entrance to a browsable presentation of the theories.

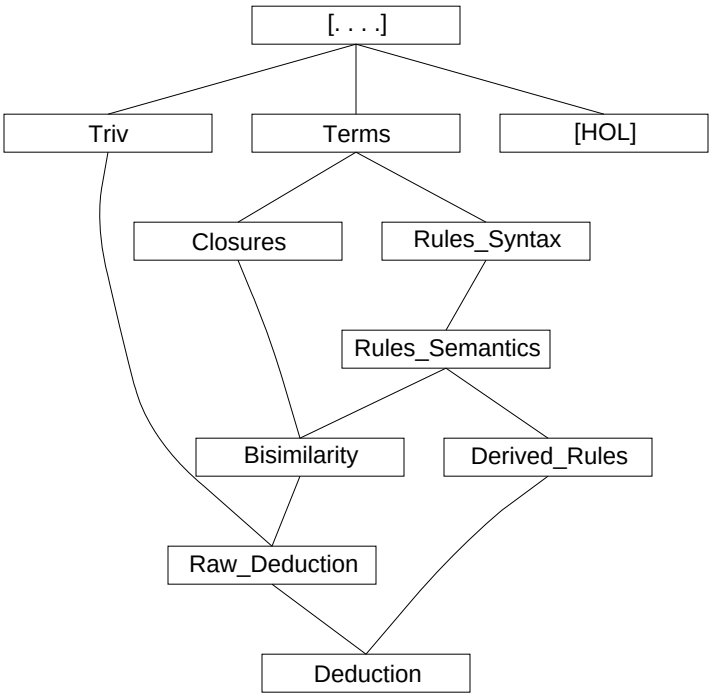

Figure 4.2: The essential part of the theory structure in Isabelle

Figures 4.2 and 4.3 present the hierarchy of our theories – the difference between Figure 4.2 and Figure 4.3 is that the former leaves out some inessential auxiliary theories, namely My_Nats and My_Lists. Here is a short description of the "essential" theories:

- Terms introduces the notion of a term and proves basic properties of substitution and occurring variables.
- Closures deals with standard closure operators on relations between terms, notably the equivalence closure, the congruence closure and the equational closure.
- Rules_Syntax introduces the notions of rule, sane rule, amenable rule, and de Simone rule, and defines the selectors "theXXs", "theXs", "theYs" etc.
- Rules_Semantics defines the operational semantics for a set of rules, i.e., the step operator.
- Bisimilarity introduces the retract functor Retr, defines the bisimilarity relation, bis, as its greatest fixpoint, and discusses various "up to" coinduction principles.

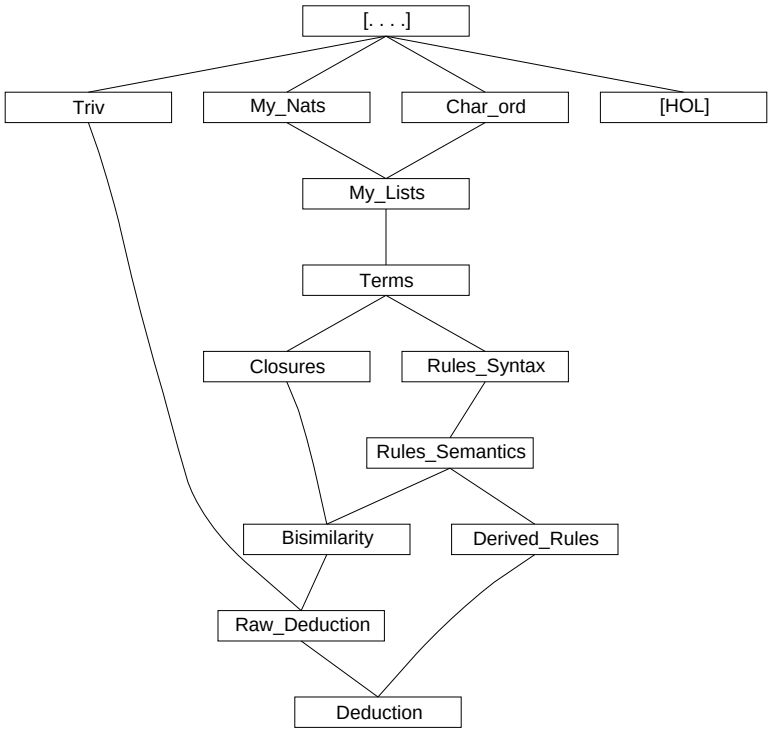

Figure 4.3: The full theory structure in Isabelle

- Derived_Rules discusses the "matched derived rule" operator, mdr.
- Raw_Deduction introduces and proves sound the raw deduction system (for bisimilarity), referred in this chapter as $\mathbf{\vdash}$.
- Deduction introduces and proves sound the deduction system for universal bisimilarity, referred in this chapter as $\vdash$.

Here are some further guidelines concerning the correspondence between the text and the scripts:

- This chapter's Section 4.2 corresponds to the theories Terms, Rules_Syntax, Rules_Semantics and Bisimilarity. In the scripts, the unspecified types **opsym**, **param** and **act** are type variables, while **var** is the type of strings of ASCII characters. Therefore, the type **term** is parametrized by **opsym** and **param**, and the type **rule** is parametrized by **opsym**, **param** and **act**. The "up-to" coinduction Theorem 4.1 from this chapter is Lemma cong_bis_coinduct in the theory Bisimilarity.
- This chapter's Section 4.3 corresponds very faithfully to the theory Raw_Deduction. As a matter of notation, the text uses for raw deduction the infix $\mathbf{\vdash}$, while the scripts call the operator "rded". Lemmas 4.3 and 4.4 and Theorem 4.2 from this chapter are Lemmas Right_Left and Left_Right and Theorem rded_sound in the scripts.
- This chapter's Section 4.4 corresponds to the theories Derived_Rules and Deduction. As a matter of notation, the text uses for deduction the infix $\vdash$, while the scripts call the deduction operator "ded". The deduction relation is slightly stronger in the scripts, since it also offers the possibility to exclude inconsistent derived rules, i.e., ones that have non-realizable side-condition – in the chapter's text, we omitted this extra twist in order to ease the presentation. Moreover, remember that in the text an equation $P \cong Q$ is merely the pair (P, Q) – in the scripts, we use the pair notation. Theorem 4.5 from this chapter is Theorem ded_sound in the theory Deduction.
- The notion of working in the context of a fixed set *Rls* of de Simone rules is captured by an Isabelle locale [73], named deSimone_Rls, used in the theories Rules_Semantics, Bisimilarity, Derived_Rules, Raw_Deduction and Deduction.

4.8 Related work

Unique fixpoint induction for CCS and its variants [86, 65, 96] is an early notion of proof-theoretic circularity for coinduction applicable to situations where circularity is *explicit* in the SOS by means of (guarded) fixpoint equations. We conjecture that unique fixpoint induction in an instance of our incremental coinduction.

We had two major sources of inspiration in this chapter. First, the idea of *circular coinduction* (CC for short) in the context of algebraic specifications. It was introduced in [55] in the behavioral specification language BOBJ [1], and then also implemented axiomatically in Isabelle under the "supervision" of the CoCASL specification language [35] and in Maude [31] as the circular coiductive prover CIRC [77, 76, 133]. A comparison of our proof system with CC is somewhat difficult to sketch, as it has to deal with different technical settings and to balance the advantages of both generality and specialization. To simplify the discussion, we shall implicitly assume a back-and-forth translation between SOS specifications and the coalgebraic and behavioral specifications required by the CC settings. Our proof system is in a sense more general and in a sense more specialized.

It is more general in that it applies to *nondeterministic processes*, not handled by CC (e.g., the running example in this chapter is not approachable in CC, not even interactively). On the other hand, CIRC, based on rewriting logic, could take advantage of the results presented here in order to extend CC with nondeterminism.[3] Also, CoCASL as a specification language has the expressive power required to deal with process algebra and nondeterminism, hence to support a version of CC for nondeterministic systems. Moreover, it is precisely the determinism of CC that allows for partial automation, admirably illustrated by CIRC. (Our formalized system, once fine-tuned into a tool, will also allow automation for deterministic, and even finitely-branching cases – see below the discussion on future work.)

It is more specialized in that the deterministic instances of our setting are more restricted than what CC can handle (in particular, e.g., deterministic lookahead, not approachable here, is unproblematic in CC). On the other hand, our powerful coinduction "up to", underneath arbitrary contexts (not supported by CC) is possible precisely because of this restriction.

Finally, our coinductive technique is presented in a logical form, as a *proof system*, like in [133], and not as an *algorithm* like in the other cited works on CC. In [133], logical form is achieved through the introduction of so-called *freezing operators*, which are and hard to justify logically – with this respect, our proof system has the advantage of "purity".[4] (Here we should also remark some less related work: circular systems in logical form were also developed in [25, 34] for first-order logic and the μ-calculus, respectively.)

The second major source of inspiration was the notion of coinduction proofs up to bisimilarity and arbitrary contexts, introduced in [37, 87] and developed in [137, 138]. This idea also appears in a general coalgebraic setting in [21] and is illustrated by extensive examples in, e.g., [136]. The convenience of performing unrestricted equational reasoning relies essentially on the "up to" coinduction principle, Theorem 4.1.

[3] Which is not to say our proof system is a minor variation of CC – nondeterminism (technically, the interplay between our rules (Split) and (Coind) from Section 4.3) represented the main difficulty in our soundness proof.

[4] In a sense, what these freezing operators do is to guard *against* coinduction up-to, not sound in general. So again, our logical system achieves convenience because of specialization.

Other related work includes frameworks for *bisimilarity of open terms* in [129, 26, 10] (also building on the seminal work from [37]), where open terms are considered universally quantified, as we do in this chapter for universal bisimilarity. Our soundness result for $\vdash$ w.r.t. universal bisimilarity, Theorem 4.5, could have been more sharply phrased: on one hand, as a soundness result w.r.t. the notion of *bisimulation under formal hypotheses* from [37, 129]; on the other, w.r.t. to the relation from [10] (which is essentially universal bisimilarity in any conservative extension of the SOS system). Finally, [154] discusses bisimilarity proofs in a mildly specialized Gentzen system for FOL. All works cited in this paragraph discuss non-incremental proof systems, where the desired bisimulation relation needs to be fed by the user.

Descriptions of more or less automatic software tools for proving bisimilarity in process algebra abound in the literature – see [72, 78] for overviews. While most of these tools are dedicated to (and optimized for) particular process algebras (and many to *finite-state* systems), ECRINS [42] is based precisely on generic process algebra in de Simone format, meaning that the results of this chapter on incremental coinduction apply directly to that setting (and, interestingly, a form of coinduction that "attempts to add more couples to the [previously specified] relation" is indicated in [42] as a direction for future research, to our knowledge not pursued so far). Finally, in Coq [23], the interaction between its general-purpose support for building proofs and its coinductive types (as illustrated, e.g., in [52]) also leads to a form of incremental coinduction whose relationship with our approach is yet to be understood.

Chapter 5

Conclusions and future work

In this final chapter we draw some conclusions about formalization aspects and discuss plans for future work.

5.1 Lessons learned from formal reasoning

HOAS is not always an alternative to FOAS. First, the obvious, often overlooked: *any* HOAS representation is based on a preliminary FOAS theory. E.g., LF is introduced in [63] employing FOAS (including α-equivalence and such[1]); and all the (informal) adequacy proofs about LF representations are FOAS proofs – if one is to make the adequacy proofs themselves formal (the only way to qualify a result obtained by HOAS as fully certified), one needs to use FOAS (as in, e.g., the formalization of LF from [151]).

Second, the explicit presence of variables in a FOAS binder such as $\mathsf{Lm}\ x\ X$, as inconvenient as it may be to manipulate, is nevertheless a fundamental operator. Thus, certain types of constructions and statements refer explicitly to this (non-injective) FOAS operator, and there is no HOAS alternative for them. For example, our motivational Problem I from Section 2.3.1 requires that the interpretation of $\mathsf{Lm}\ x\ X$ in a semantic domain be performed by assigning x different values in the given valuations. This is an irreducible phenomenon, not only mathematically, but also pragmatically: syntax is available to its users (e.g., the programmers) by means of FOAS binding operators, since of course particular names of variables are chosen when writing concrete programs or proofs in a formal system such as a programming language or a theorem prover; and any denotational-semantics map has to account for this.

These being said, it is true that HOAS, when it works, it rocks. Thus, when irreducibly first-order aspects are not important in themselves, but only as auxiliaries in achieving a different goal, HOAS alternatives can simplify and clean-up reasoning – this was the case of our proof of Strong Normalization for System F from Section 3.5, where this very issue of semantic interpretation was bypassed. (And, of course, there are many other illustrations of

[1] Which is not to say that α-equivalence has to be employed in the basic definition. But the first-order operators have to be there.

the benefits of HOAS in the literature.)

The above are the main reasons for our mixed approach to syntax with bindings, which could be summarized as follows: FOAS when necessary, HOAS when possible. The fact that HOAS comes definitionally "on top of FOAS" makes it easy to switch between the two.

Formalizing patiently avoids formalizing painfully.[2] When starting the work reported here, the author of this dissertation was a pure "pen and paper" mathematician, and regarded formalization as only the "necessary evil" needed to illustrate or apply some ideas. Eventually though, formalization revealed itself as a great tool for *better understanding* the involved concepts. The only prerequisite for this, which can make the difference between formalization as pain and formalization as pleasure, is, in our opinion, observing "religiously" the following principle:

When a concept (no matter how simple) is formalized, it should be related, by means of lemmas, to *each and every other concept in its vicinity*, in all possible obvious ways. E.g., assuming parallel substitution is defined in a context where operators such as swapping, freshness etc. already exist, all the simple lemmas pertaining to the interaction of substitution with these operators (commutation, preservation, special interaction with the syntactic constructs in the presence of freshness, interaction with environment update, etc.) should be proved.

This may launch combinatorial mini-explosions of lemmas and thus may seem like a delay in the development, but we have noticed a phenomenon rather curious by its pervasiveness: every time we decided to leave such an obvious connection lemma unstated (not anticipating, at that point, any future use of it), eventually we have discovered we need that fact. And having the hard combinatorial work done *early*, when proving conceptually trivial facts, is way better[3] than facing it later, in more complex proof situations.[4] For the latter situations, it is crucial to allow ourselves following higher-level ideas, as in informal mathematical proofs. In fact, by observing the mentioned principle, and adding many basic facts into the working background by means of simplification rules, we were often able to proceed even more "informally" than one would expect from a rigorous pen-and-paper proof.[5] For example, many informal proofs by induction with several occurring cases, proceed with an "we only prove the interesting case (or cases)" disclaimer. This is hardly accepted as

[2]The whole discussion here is essentially a restatement of the classic benefits of modularity – however, there are some specific nuances.

[3]From all points of view: time, space, clarity, maintainability.

[4]An analogy that comes to mind is category theory. Contrary to what it is sometimes claimed, an approach based on category theory to solve a concrete problem is *not* avoiding the consideration of low-level details. It just handles these details, typically by means of tons of routine verifications, in an early stage, e.g., when organizing the given concrete items as categories, functors, natural transformations etc.. The reward is an abstract, "detail-free" setting later, when it is most needed: in the non-trivial phase of the problem-solving task.

[5]Of course, the advanced facilities of Isabelle-Isar [101] did have a say here.

a rigorous pen-and-paper proof, but Isabelle can be convinced that the chosen cases are indeed the only interesting ones (full automation discharging the rest). Again, the key here is some preliminary (admittedly, tedious) work at customizing Isabelle to accept "without questioning" roughly everything we consider obvious on a given topic.

5.2 Future work

In this dissertation, we have pursued two separate themes:
- an inductive (and recursive) theme, of representing and reasoning about syntax with bindings;
- a coinductive theme, of representing and reasoning about behavior of processes.

These themes correspond to the standard duality between the (static) structure and (dynamic) behavior of programming languages. Important aspects our approaches to these two themes have in common are (for most of the work) the generality of the setting:
- arbitrary syntax with bindings in the first case;
- arbitrary signature for processes and arbitrary set of SOS rules in a general format, in the second.

This generality is reflected in our corresponding Isabelle formalizations.

We plan to eventually integrate induction and coinduction in a single comprehensive Isabelle package where one can specify and reason about λ-calculi, process calculi and programming languages. There is long way to go, but the outcome is in sight. Next we describe the steps we plan to take towards this goal.

Generalization of the HOAS constructios. The constructions and results from Section 3.3 can be straightforwardly generalized to an arbitrary many-sorted syntax with bindings. Moreover, the constructions and adequacy proofs from Section 3.4 seem to work for a large class of inductively defined inference systems in whose clauses the migration of variables between scopes satisfies a few general conditions, allowing the sound application of transformations (I)-(V) discussed in Section 3.4. We are currently working on determining such suitably general conditions.

Full automation of the FOAS and HOAS constructions. Although most of our results have been formalized in a general setting, we have not yet taken full advantage of the ample possibilities for automatically building the involved FOAS and HOAS apparatuses. We shall implement (the general versions of) all our results presented in Chapters 2 and 3 as a definitional package in Isabelle/HOL. Our system will require the user to give a *binding signature* and a number of *inference system specifications* on the terms of this signatures (for various desired relations: typing, operational semantics, etc.). From the binding signature,

the system will produce the terms (one Isabelle type of terms for each syntactic category), as well as all the standard operators on them (substitution, free variables etc.) and prove the standard lemmas about them. Moreover, a recursor shall be provided, based on our work from Chapter 2. (Thus, essentially, all the FOAS results we have gathered "by hand" in the theory L from [121] for the syntax of λ-calculus shall be provided automatically for the indicated syntax.)

From the inference system specifications, the system will produce the actual inductive definitions of the intended relations. Then the system will construct, along the lines of Chapter 3, the HOAS view of syntax (defining new higher-order operators on terms and proving their properties) and the representation of inference, which will be automatically proved adequate. General versions of the propositions in Sections 3.3 and 3.4 shall also be proved (automatically). All in all, based on a very compact input from the user, our system will produce:

- **(i)** the intended object system with all its FOAS constructions;
- **(ii)** a HOAS representation formally certified as adequate.

Merging the formalizations of induction and coinduction. The missing link with the coinductive work lays in the fact that currently our syntax for processes does not involve any bindings. However, our process algebra formalization is quite modular, so that switching to a syntax with bindings for processes (and thus capturing important cases such as the π-calculus and real concurrent programming languages) should be in principle realizable without major changes to the coinductive engine. This way, the package will also include:

- **(iii)** a (customized) proof system for coinduction automatically inferable from an SOS specification of the intended behavior.

There are of course several design decisions that need to be made towards a coherent inductive-coinductive package – e.g., one needs to establish what counts for a system for which it makes sense to extract a coinductive notion of behavior from an inductive SOS specification.

Relaxing bisimilarity. Relaxing the notion of bisimilarity to variants that consider issues such as *divergence*, *asynchronous communication*, *fairness* [48] and combinations of these such as the *guarantee of message delivery* [32, 11] is an important step for being able to model behavior of real programming languages and systems. While traditionally fairness is connected to trace-based rather than bisimilarity-based semantics, our discussion in Section 4.5.2 about weak bisimilarity shows that modular treatment of this complex equivalence is actually achieved by a mixture of traces and coinduction. Also, recent work [98, 100, 99] has demonstrated that weak-bisimilarity-based process algebra is closer to programming languages than one may initially expect. Moreover, asynchrony can be achieved by certain

syntactic restrictions on processes [138].

References

[1] BOBJ. http://cseweb.ucsd.edu/groups/tatami/bobj.

[2] Delphin. http://cs-www.cs.yale.edu/homes/carsten/delphin.

[3] LEGO. http://www.dcs.ed.ac.uk/home/lego.

[4] The Abella Theorem prover, 2009. http://abella.cs.umn.edu/.

[5] The POPLmark challenge, 2009. http://fling-l.seas.upenn.edu/ plclub/cgi-bin/poplmark/.

[6] The Twelf Project, 2009. http://twelf.plparty.org/.

[7] The HOL4 Theorem prover, 2010. http://hol.sourceforge.net/.

[8] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.

[9] S. Abramsky and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Inf. Comput.*, 105(2):159–267, 1993.

[10] L. Aceto, M. Cimini, and A. Ingólfsdóttir. A bisimulation-based method for proving the validity of equations in gsos languages. *CoRR*, abs/1002.2864, 2010.

[11] G. Agha. *Actors: a model of concurrent computation in distributed systems.* MIT Press, Cambridge, MA, USA, 1986.

[12] T. Altenkirch. A formalization of the strong normalization proof for System F in LEGO. In *TLCA*, pages 13–28, 1993.

[13] S. Ambler, R. L. Crole, and A. Momigliano. Combining Higher Order Abstract Syntax with tactical theorem proving and (co)induction. In *TPHOLs*, pages 13–30, 2002.

[14] S. J. Ambler, R. L. Crole, and A. Momigliano. A definitional approach to primitive recursion over Higher Order Abstract Syntax. In *MERLIN*, 2003.

[15] A. Avron, F. Honsell, I. A. Mason, and R. Pollack. Using typed λ-calculus to implement formal systems on a machine. *J. of Aut. Reasoning*, 9(3):309–354, 1992.

[16] E. Badouel and P. Darondeau. On guarded recursion. *Theor. Comput. Sci.*, 82:403–408, 1991.

[17] J. Baeten and W. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[18] H. Barendregt. Introduction to generalized type systems. *J. Funct. Program.*, 1(2):125–154, 1991.

[19] H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*. Oxford University Press, 1992.

[20] H. P. Barendregt. *The Lambda Calculus*. North-Holland, 1984.

[21] F. Bartels. Generalised coinduction. *Math. Struct. Comp. Sci.*, 13(2):321–348, 2003.

[22] M. Berger, K. Honda, and N. Yoshida. Genericity and the pi-calculus. *Acta Inform.*, 42(2):83–141, 2005.

[23] Y. Bertot and P. Casteran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

[24] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995.

[25] J. Brotherston. Cyclic proofs for first-order logic with inductive definitions. In *TABLEAUX'05*, pages 78–92, 2005.

[26] R. Bruni, D. de Frutos-Escrig, N. Martí-Oliet, and U. Montanari. Bisimilarity congruences for open terms and term graphs via tile logic. In *CONCUR*, pages 259–274, 2000.

[27] V. Capretta and A. P. Felty. Combining de Bruijn indices and higher-order abstract syntax in Coq. In *TYPES*, pages 63–77, 2006.

[28] M. Cerioli and J. Meseguer. May I borrow your logic? (Transporting logical structures along maps). *Theoretical Computer Science*, 173(2):311–347, 1997.

[29] C. Chen and H. Xi. Combining programming with theorem proving. In *ICFP*, pages 66–77, 2005.

[30] A. J. Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *ICFP*, pages 143–156, 2008.

[31] M. Clavel, F. J. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. The Maude system. In P. Narendran and M. Rusinowitch, editors, *Proceedings of the 10th International Conference on Rewriting Techniques and Applications (RTA-99)*, volume 1631 of *Lecture Notes in Computer Science*, pages 240–243, Trento, Italy, July 1999. Springer-Verlag. System Description.

[32] W. Clinger. Foundations of actor semantics. Mathematics Doctoral Dissertation. MIT, 1981.

[33] P.-L. Curien. Categorical combinators. *Information and Control*, 69(1-3):188–254, 1986.

[34] M. Dam and D. Gurov. μ-calculus with explicit points and approximations. *J. Log. Comput.*, 12(2):255–269, 2002.

[35] T. M. Daniel Hausmann and L. Schrder. Iterative circular coinduction for cocasl in isabelle/hol. In *Fundamental Approaches to Software Engineering 2005*, Lecture Notes in Computer Science, page pp. 341356, 2005.

[36] N. de Bruijn. λ-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indag. Math*, 34(5):381–392, 1972.

[37] R. de Simone. Higher-level synchronizing devices in meije-sccs. *Theor. Comput. Sci.*, 37:245–267, 1985.

[38] J. Despeyroux, A. P. Felty, and A. Hirschowitz. Higher-order abstract syntax in Coq. In *TLCA*, pages 124–138, 1995.

[39] J. Despeyroux and A. Hirschowitz. Higher-Order Abstract Syntax with induction in Coq. In *LPAR*, pages 159–173, 1994.

[40] J. Despeyroux and P. Leleu. Recursion over objects of functional type. *Mathematical Structures in Computer Science*, 11(4):555–572, 2001.

[41] K. Donnelly and H. Xi. A formalization of strong normalization for simply-typed lambda-calculus and system F. *Electron. Notes Theor. Comput. Sci.*, 174(5):109–125, 2007.

[42] G. Doumenc, E. Madelaine, and R. de Simone. Proving process calculi translations in ECRINS: The pureLOTOS $\rightarrow$ MEIJE example. Technical Report RR1192, INRIA, 1990. http://hal.archives-ouvertes.fr/inria-00075367/en/.

[43] A. P. Felty and A. Momigliano. Hybrid: A definitional two-level approach to reasoning with Higher-Order Abstract Syntax. *CoRR*, abs/0811.4367, 2008.

[44] A. P. Felty and A. Momigliano. Reasoning with hypothetical judgments and open terms in hybrid. In *PPDP*, pages 83–92, 2009.

[45] A. P. Felty and B. Pientka. Reasoning with higher-order abstract syntax and contexts: A comparison. In *ITP*, pages 227–242, 2010.

[46] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding (extended abstract). In *LICS*, pages 193–202, 1999.

[47] M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the π-calculus. In *LICS*, pages 43–54, 1996.

[48] N. Francez. *Fairness*. Springer-Verlag, 1986.

[49] M. J. Gabbay. A theory of inductive definitions with α-equivalence. Ph.D. thesis. University of Cambridge, 2001.

[50] A. Gacek, D. Miller, and G. Nadathur. Combining generic judgments with recursive definitions. In F. Pfenning, editor, *LICS*, pages 33–44, June 2008.

[51] J. Gallier. On Girard's candidats de reductibilite. In *Logic and Computer Science*, pages 123–203. Academic Press, 1990.

[52] E. Giménez. An application of co-inductive types in Coq: Verification of the alternating bit protocol. In *TYPES'95*, pages 135–152, 1995.

[53] J.-Y. Girard. Une extension de l'interpretation de Gödel a l'analyse, et son application a l'elimination des coupure dans l'analyse et la theorie des types. In *2nd Scandinavian Logic Symposium*, pages 63–92, 1971.

[54] J.-Y. Girard. *Proofs and Types*. Cambridge University Press, 1989.

[55] J. Goguen, K. Lin, and G. Roşu. Circular coinductive rewriting. In *Proceedings of Automated Software Engineering 2000*, pages 123–131. IEEE, 2000.

[56] J. Goguen and G. Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, August 2000.

[57] A. D. Gordon. A mechanisation of name-carrying syntax up to alpha-conversion. In *HUG '93: Proceedings of the 6th International Workshop on Higher Order Logic Theorem Proving and its Applications*, pages 413–425, 1994.

[58] A. D. Gordon and T. F. Melham. Five axioms of alpha-conversion. In *TPHOLs '96: Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics*, pages 173–190, 1996.

[59] J. F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Inf. Comput.*, 100(2):202–260, 1992.

[60] C. A. Gunter. *Semantics of Programming Languages. Structures and Techniques*. The MIT Press, 1992.

[61] E. L. Gunter, C. J. Osborn, and A. Popescu. Theory support for weak Higher Order Abstract Syntax in Isabelle/HOL. In *LFMTP*, pages 12–20, 2009.

[62] J. V. Guttag and J. J. Horning. The algebraic specification of abstract data types. *Acta Inf.*, 10:27–52, 1978.

[63] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *LICS*, pages 194–204. IEEE, Computer Society Press, 1987.

[64] R. Harper and D. R. Licata. Mechanizing metatheory in a logical framework. *J. Funct. Program.*, 17(4-5):613–673, 2007.

[65] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. *Formal Asp. Comput.*, 8(4):379–407, 1996.

[66] J. Hickey, A. Nogin, X. Yu, and A. Kopylov. Mechanized meta-reasoning using a hybrid HOAS/de Bruijn representation and reflection. In *ICFP*, pages 172–183, 2006.

[67] M. Hofmann. Semantical analysis of higher-order abstract syntax. In *LICS*, page 204, 1999.

[68] F. Honsell, M. Miculan, and I. Scagnetto. An axiomatic approach to metareasoning on nominal algebras in HOAS. In *ICALP*, pages 963–978, 2001.

[69] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124(2), 1996.

[70] D. J. Howe. Higher-order abstract syntax in classical higher-order logic. In *LFMTP*, pages 1–11, 2009.

[71] G. P. Huet and B. Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Inf.*, 11:31–55, 1978.

[72] P. Inverardi and C. Priami. Automatic verification of distributed systems: The process algebra approach. *Formal Methods in System Design*, 8(1):7–38, 1996.

[73] F. Kammüller, M. Wenzel, and L. C. Paulson. Locales - a sectioning concept for Isabelle. In *TPHOLs'99*, pages 149–166, 1999.

[74] J. L. Krivine. *Lambda-calculus, types and models.* Ellis Horwood, 1993.

[75] R. Loader. Normalization by calculation. Unpublished note, 1995. http://homepages.ihug.co.nz/ suckfish/papers/normal.pdf.

[76] D. Lucanu, E.-I. Goriac, G. Caltais, and G. Roşu. CIRC: A behavioral verification tool based on circular coinduction. In *CALCO'09*, pages 433–442, 2009.

[77] D. Lucanu and G. Rosu. CIRC: A circular coinductive prover. In *CALCO'07*, volume 4624 of *LNCS*, pages 372 – 378, 2007.

[78] E. Madelaine. Verification tools from the CONCUR project. http://www-sop.inria.fr/meije/papers/concur-tools.

[79] N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. *Electr. Notes Theor. Comput. Sci.*, 4, 1996.

[80] R. McDowell and D. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic*, 3(1):80–136, 2002.

[81] R. C. McDowell. *Reasoning in a logic with definitions and induction.* PhD thesis, University of Pennsylvania, 1997.

[82] J. McKinna and R. Pollack. Pure type systems formalized. In *TLCA*, pages 289–305, 1993.

[83] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, pages 73–155, 1992.

[84] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Ann. Pure Appl. Logic*, 51(1-2):125–157, 1991.

[85] D. Miller and A. Tiu. A proof theory for generic judgments. *ACM Transactions on Computational Logic*, 6(4):749–783, 2005.

[86] R. Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984.

[87] R. Milner. *Communication and concurrency*. Prentice Hall, 1989.

[88] R. Milner. *Communicating and mobile systems: the π-calculus*. Cambridge, 2001.

[89] J. C. Mitchell. A type-inference approach to reduction properties and semantics of polymorphic expressions (summary). In *LISP and Functional Programming*, pages 308–319, 1986.

[90] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[91] J. C. Mitchell and A. R. Meyer. Second-order logical relations (extended abstract). In *CLP*, pages 225–236, 1985.

[92] A. Momigliano and S. Ambler. Multi-level meta-reasoning with higher-order abstract syntax. In *FoSSaCS*, pages 375–391, 2003.

[93] A. Momigliano, S. Ambler, and R. L. Crole. A hybrid encoding of howe's method for establishing congruence of bisimilarity. *Electr. Notes Theor. Comput. Sci.*, 70(2), 2002.

[94] A. Momigliano, S. J. Ambler, and R. L. Crole. A comparison of formalizations of the meta-theory of a language with variable bindings in isabelle. Technical report, Supplemental Proceedings of TPHOL'01, 2001.

[95] A. Momigliano, A. J. Martin, and A. P. Felty. Two-level Hybrid: A system for reasoning using Higher-Order Abstract Syntax. *Electron. Notes Theor. Comput. Sci.*, 196:85–93, 2008.

[96] R. Monroy, A. Bundy, and I. Green. On process equivalence = equation solving in ccs. *J. Autom. Reasoning*, 43(1):53–80, 2009.

[97] M. R. Mousavi, M. A. Reniers, and J. F. Groote. Sos formats and meta-theory: 20 years after. *Theor. Comput. Sci.*, 373(3):238–272, 2007.

[98] K. Nakata and T. Uustalu. Trace-based coinductive operational semantics for while. In *TPHOLs*, pages 375–390, 2009.

[99] K. Nakata and T. Uustalu. A hoare logic for the coinductive trace-based big-step semantics of while. In *ESOP*, pages 488–506, 2010.

[100] K. Nakata and T. Uustalu. Resumptions, weak bisimilarity and big-step semantics for while with interactive i/o: An exercise in mixed induction-coinduction. *CoRR*, abs/1008.2112, 2010.

[101] T. Nipkow. Structured Proofs in Isar/HOL. In *TYPES*, pages 259–278, 2003.

[102] T. Nipkow and L. C. Paulson. Proof pearl: Defining functions over finite sets. In *TPHOLs*, pages 385–396, 2005.

[103] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer, 2002.

[104] M. Norrish. Recursive function definition for types with binders. In *TPHOLs*, pages 241–256, 2004.

[105] M. Norrish. Mechanising lambda-calculus using a classical first order theory of terms with permutations. *Higher-Order and Symbolic Computation*, 19(2-3):169–195, 2006.

[106] M. Norrish and R. Vestergaard. Proof pearl: De Bruijn terms really do work. In *TPHOLs*, pages 207–222, 2007.

[107] H. Ohtsuka. A proof of the substitution lemma in de Bruijn's notation. *Inf. Process. Lett.*, 46(2):63–66, 1993.

[108] L. C. Paulson. The foundation of a generic theorem prover. *J. Autom. Reason.*, 5(3), 1989.

[109] F. Pfenning. Logical frameworks. In *Handbook of Automated Reasoning*. Elsevier Science, 1999.

[110] F. Pfenning. Logical frameworks - a brief introduction. In *Paris Colloqvium on Programming*, volume 62 of *NATO Science Series II*, pages 137–166. Kluwer Academic Publishers, 2002.

[111] F. Pfenning and C. Elliot. Higher-order abstract syntax. In *PLDI*, pages 199–208, 1988.

[112] B. Pientka. Proof pearl: The power of higher-order encodings in the logical framework lf. In *TPHOLs*, pages 246–261, 2007.

[113] B. Pientka. Beluga: Programming with dependent types, contextual data, and contexts. In *FLOPS*, pages 1–12, 2010.

[114] B. C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[115] A. M. Pitts. Nominal logic: A first order theory of names and binding. In *TACS*, pages 219–242, 2001.

[116] A. M. Pitts. Alpha-structural recursion and induction. *J. ACM*, 53(3), 2006.

[117] G. D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[118] R. Pollack and M. Sato. A canonical locally named representation of binding. To appear in Journal of Automated Reasoning.

[119] A. Popescu. HOAS on top of FOAS formalized in Isabelle/HOL. Technical report http://hdl.handle.net/2142/15449. Department of Computer Science, University of Illinois at Urbana-Champaign, 2010.

[120] A. Popescu. The Isabelle formalization of a general theory of syntax with bindings. Tech. Rep., Univ. of Illinois at Urbana-Champaign, 2010. http://hdl.handle.net/2142/17423.

[121] A. Popescu. The Isabelle formalization of a general theory of syntax with bindings, with λ-calculus case studies included. Tech. Rep., Univ. of Illinois at Urbana-Champaign, 2010. http://hdl.handle.net/2142/17424.

[122] A. Popescu. The Isabelle formalization of an incremental coniductive proof system. Tech. Rep., Univ. of Illinois at Urbana-Champaign, 2010. http://hdl.handle.net/2142/14857.

[123] A. Popescu. Weak bisimilarity coalgebraically. In *CALCO'09*, pages 157–172, 2009.

[124] A. Popescu and E. Gunter. Incremental pattern-based coinduction for process algebra and its Isabelle formalization. Technical report http://hdl.handle.net/2142/14858. Department of Computer Science, University of Illinois at Urbana-Champaign, 2010.

[125] A. Popescu and E. L. Gunter. Incremental pattern-based coinduction for process algebra and its Isabelle formalization. In *FOSSACS'10*, 2010.

[126] A. Popescu, E. L. Gunter, and C. J. Osborn. Strong normalization of System F by HOAS on top of FOAS. In *LICS*, pages 31–40, 2010.

[127] A. Popescu and G. Roşu. Term-generic logic. In *WADT*, pages 290–307, 2008.

[128] A. Popescu and G. Roşu. Term-generic logic. Technical Report UIUCDCS-R-2009-3027. Department of Computer Science, University of Illinois at Urbana-Champaign, 2009.

[129] A. Rensink. Bisimilarity of open terms. *Inf. Comput.*, 156(1-2):345–385, 2000.

[130] J. Reynolds. Towards a theory of type structure. In *Paris Colloqvium on Programming*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer-Verlag, 1974.

[131] C. Röckl and D. Hirschkoff. A fully adequate shallow embedding of the [pi]-calculus in Isabelle/HOL with mechanized syntax analysis. *J. Funct. Program.*, 13(2):415–451, 2003.

[132] G. Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000. `http://ase.arc.nasa.gov/grosu/phd-thesis.ps`.

[133] G. Roşu and D. Lucanu. Circular coinduction: A proof theoretical foundation. In *CALCO'09*, pages 127–144, 2009.

[134] J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.

[135] J. J. M. M. Rutten. Processes as terms: Non-well-founded models for bisimulation. *Math. Struct. Comp. Sci.*, 2(3):257–275, 1992.

[136] J. J. M. M. Rutten. Elements of stream calculus (an extensive exercise in coinduction). *Electr. Notes Theor. Comput. Sci.*, 45, 2001.

[137] D. Sangiorgi. On the bisimulation proof method. *Math. Struct. Comp. Sci.*, 8(5):447–479, 1998.

[138] D. Sangiorgi and D. Walker. *The π-calculus. A theory of mobile processes*. Cambridge, 2001.

[139] M. Sato and R. Pollack. External and internal syntax of the lambda-calculus. *Journal of Symbolic Computation*, 45:598–616, 2010.

[140] C. Schurmann. *Automating the meta-theory of deductive systems*. PhD thesis, Carnegie Mellon University, 2000.

[141] C. Schurmann, J. Despeyroux, and F. Pfenning. Primitive recursion for higher-order abstract syntax. *Theor. Comput. Sci.*, 266(1-2):1–57, 2001.

[142] C. Schurmann and F. Pfenning. Automated theorem proving in a simple meta-logic for LF. In *CADE*, pages 286–300, 1998.

[143] R. Statman. Logical relations and the typed lambda-calculus. *Information and Control*, 65(2/3):85–97, 1985.

[144] W. Tait. A realizability interpretation of the theory of species. In *Logic Colloquium*, pages 240–251. Springer, 1975.

[145] M. Takahashi. Parallel reductions in lambda-calculus. *Inf. Comput.*, 118(1):120–127, 1995.

[146] A. Tiu. *A Logical Framework for Reasoning about Logical Specifications*. PhD thesis, Penn State University, 2004.

[147] D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 280–291, Warsaw, Poland, 29 June–2 July 1997. Institute of Electrical and Electronics Engineers Computer Society Press.

[148] C. Urban. Nominal techniques in Isabelle/HOL. *J. Autom. Reason.*, 40(4):327–356, 2008.

[149] C. Urban and S. Berghofer. A recursion combinator for nominal datatypes implemented in isabelle/hol. In *IJCAR*, pages 498–512, 2006.

[150] C. Urban, S. Berghofer, and M. Norrish. Barendregt's variable convention in rule inductions. In *CADE*, pages 35–50, 2007.

[151] C. Urban, J. Cheney, and S. Berghofer. Mechanizing the metatheory of lf. In *LICS*, pages 45–56, 2008.

[152] C. Urban and M. Norrish. A formal treatment of the Barendregt variable convention in rule inductions. In *MERLIN*, pages 25–32, 2005.

[153] C. Urban and C. Tasson. Nominal techniques in isabelle/hol. In *CADE*, pages 38–53, 2005.

[154] M. van Weerdenburg. Automating soundness proofs. *Electr. Notes Theor. Comput. Sci.*, 229(4):107–118, 2009.

[155] M. Wenzel, L. C. Paulson, and T. Nipkow. The isabelle framework. In *TPHOLs*, pages 33–38, 2008.