Roberto Bruni, Ugo Montanari

Models of Computation

– Monograph –

April 20, 2016

DRAFT

Springer

Mathematical reasoning may be regarded rather schematically as the exercise of a combination of two facilities, which we may call intuition and ingenuity.

Alan Turing[1]

DRAFT

[1] The purpose of ordinal logics (from Systems of Logic Based on Ordinals), Proceedings of the London Mathematical Society, series 2, vol. 45, 1939.

Preface

The origins of this book lie their roots on more than 15 years of teaching a course on formal semantics to graduate Computer Science to students in Pisa, originally called *Fondamenti dell'Informatica: Semantica* (*Foundations of Computer Science: Semantics*) and covering models for imperative, functional and concurrent programming. It later evolved to *Tecniche di Specifica e Dimostrazione* (*Techniques for Specifications and Proofs*) and finally to the currently running *Models of Computation*, where additional material on probabilistic models is included.

The objective of this book, as well as of the above courses, is to present different *models of computation* and their basic *programming paradigms*, together with their mathematical descriptions, both *concrete* and *abstract*. Each model is accompanied by some relevant formal techniques for reasoning on it and for proving some properties.

To this aim, we follow a rigorous approach to the definition of the *syntax*, the *typing* discipline and the *semantics* of the paradigms we present, i.e., the way in which well-formed programs are written, ill-typed programs are discarded and the way in which the meaning of well-typed programs is unambiguously defined, respectively. In doing so, we focus on basic proof techniques and do not address more advanced topics in detail, for which classical references to the literature are given instead.

After the introductory material (Part I), where we fix some notation and present some basic concepts such as term signatures, proof systems with axioms and inference rules, Horn clauses, unification and goal-driven derivations, the book is divided in four main parts (Parts II-V), according to the different styles of the models we consider:

IMP: imperative models, where we apply various incarnations of well-founded induction and introduce λ-notation and concepts like structural recursion, program equivalence, compositionality, completeness and correctness, and also complete partial orders, continuous functions, fixpoint theory;

HOFL: higher-order functional models, where we study the role of type systems, the main concepts from domain theory and the distinction between lazy and eager evaluation;

DRAFT

CCS, π: concurrent, non-deterministic and interactive models, where, starting from operational semantics based on labelled transition systems, we introduce the notions of bisimulation equivalences and observational congruences, and overview some approaches to name mobility, and temporal and modal logics system specifications;

PEPA: probabilistic/stochastic models, where we exploit the theory of Markov chains and of probabilistic reactive and generative systems to address quantitative analysis of, possibly concurrent, systems.

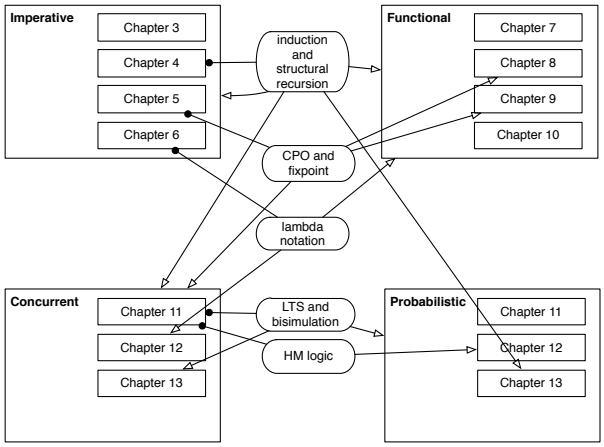
Each of the above models can be studied in separation from the others, but previous parts introduce a body of notions and techniques that are also applied and extended in later parts.

Parts I and II cover the essential, classic topics of a course on formal semantics.

Part III introduces some basic material on process algebraic models and temporal and modal logic for the specification and verification of concurrent and mobile systems. CCS is presented in good detail, while the theory of temporal and modal logic, as well as π-calculus, are just overviewed. The material in Part III can be used in conjunction with other textbooks, e.g., on model checking or π-calculus, in the context of a more advanced course on the formal modelling of distributed systems.

Part IV outlines the modelling of probabilistic and stochastic systems and their quantitative analysis with tools like PEPA. It poses the basis for a more advanced course on quantitative analysis of sequential and interleaving systems.

The diagram that highlights the main dependencies is represented below:



The diagram contains a squared box for each chapter / part and a rounded-corner box for each subject: a line with a filled-circle end joins a subject to the chapter where it is introduced, while a line with an arrow end links a subject to a chapter or part where it is used. In short:

Induction and recursion: various principles of induction and the concept of structural recursion are introduced in Chapter 4 and used extensively in all subsequent chapters.

CPO and fixpoint: the notion of complete partial order and fixpoint computation are first presented in Chapter 5. They provide the basis for defining the denotational semantics of IMP and HOFL. In the case of HOFL, a general theory of product and functional domains is also introduced (Chapter 8). The notion of fixpoint is also used to define a particular form of equivalence for concurrent and probabilistic systems, called bisimilarity, and to define the semantics of modal logic formulas.

Lambda-notation: λ-notation is a useful syntax for managing anonymous functions. It is introduced in Chapter 6 and used extensively in Part III.

LTS and bisimulation: Labelled transition systems are introduced in Chapter 11 to define the operational semantics of CCS in terms of the interactions performed. They are then extended to deal with name mobility in Chapter 13 and with probabilities in Part V. A bisimulation is a relation over the states of an LTS that is closed under the execution of transitions. The before mentioned bisimilarity is the coarsest bisimulation relation. Various forms of bisimulation are studied in Part IV and V.

HM-logic: Hennessy-Milner logic is the logic counterpart of bisimilarity: two state are bisimilar if and only if they satisfy the same set of HM-logic formulas. In the context of probabilistic system, the approach is extended to Larsen-Skou logic in Chapter 15.

Each chapter of the book is concluded by a list of exercises that span over the main techniques introduced in that chapter. Solutions to selected exercises are collected at the end of the book.

Pisa, *Roberto Bruni*
February 2016 *Ugo Montanari*

DRAFT

Acknowledgements

We want to thank our friend and colleague Pierpaolo Degano for encouraging us to prepare this book and submit it to the EATCS monograph series. We thank Ronan Nugent and all the people at Springer for their editorial work. We acknowledge all the students of the course on *Models of Computation (MOD)* in Pisa for helping us to refine the presentation of the material in the book and to eliminate many typos and shortcomings from preliminary versions of this text. Last but not least, we thank Lorenzo Galeotti, Andrea Cimino, Lorenzo Muti, Gianmarco Saba, Marco Stronati, former students of the course on *Models of Computation*, who helped us with the LaTeX preparation of preliminary versions of this book, in the form of lecture notes.

DRAFT

Contents

DRAFT

DRAFT

DRAFT

DRAFT

DRAFT

DRAFT

Acronyms

$\sim$	operational equivalence in IMP (see Definition 3.3)
$\equiv_{den}$	denotational equivalence in HOFL (see Definition 10.4)
$\equiv_{op}$	operational equivalence in HOFL (see Definition 10.3)
$\simeq$	CCS strong bisimilarity (see Definition 11.5)
$\approx$	CCS weak bisimilarity (see Definition 11.16)
$\cong$	CCS weak observational congruence (see Section 11.8.2)
$\approx_d$	CCS dynamic bisimilarity (see Definition 11.17)
$\overset{\circ}{\sim}_E$	π-calculus early bisimilarity (see Definition 13.3)
$\overset{\circ}{\sim}_L$	π-calculus late bisimilarity (see Definition 13.4)
$\sim_E$	π-calculus strong early full bisimilarity (see Section 13.5.3)
$\sim_L$	π-calculus strong late full bisimilarity (see Section 13.5.3)
$\overset{\bullet}{\approx}_E$	π-calculus weak early bisimilarity (see Section 13.5.4)
$\overset{\bullet}{\approx}_L$	π-calculus weak late bisimilarity (see Section 13.5.4)
$\mathscr{A}$	interpretation function for the denotational semantics of IMP arithmetic expressions (see Section 6.2.1)
ack	Ackermann function (see Example 4.18)
Aexp	set of IMP arithmetic expressions (see Chapter 3)
$\mathscr{B}$	interpretation function for the denotational semantics of IMP boolean expressions (see Section 6.2.2)
Bexp	set of IMP boolean expressions (see Chapter 3)
$\mathbb{B}$	set of booleans
$\mathscr{C}$	interpretation function for the denotational semantics of IMP commands (see Section 6.2.3)
CCS	Calculus of Communicating Systems (see Chapter 11)
Com	set of IMP commands (see Chapter 3)
CPO	Complete Partial Order (see Definition 5.11)
$\text{CPO}_\perp$	Complete Partial Order with bottom (see Definition 5.12)
CSP	Communicating Sequential Processes (see Section 16.2)
CTL	Computation Tree Logic (see Section 12.1.2)
CTMC	Continuous Time Markov Chain (see Definition 14.15)

DRAFT

DTMC	Discrete Time Markov Chain (see Definition 14.14)
Env	set of HOFL environments (see Chapter 9)
fix	(least) fixpoint (see Definition 5.2.2)
FIX	(greatest) fixpoint
gcd	greatest common divisor
HML	Hennessy-Milner modal Logic (see Section 11.6)
HM-Logic	Hennessy-Milner modal Logic (see Section 11.6)
HOFL	A Higher-Order Functional Language (see Chapter 7)
IMP	A simple IMPerative language (see Chapter 3)
int	integer type in HOFL (see Definition 7.2)
Loc	set of locations (see Chapter 3)
LTL	Linear Temporal Logic (see Section 12.1.1)
LTS	Labelled Transition System (see Definition 11.2)
lub	least upper bound (see Definition 5.7)
$\mathbb{N}$	set of natural numbers
$\mathscr{P}$	set of closed CCS processes (see Definition 11.1)
PEPA	Performance Evaluation Process Algebra (see Chapter 16)
Pf	set of partial functions on natural numbers (see Example 5.13)
PI	set of partial injective functions on natural numbers (see Problem 5.12)
PO	Partial Order (see Definition 5.1)
PTS	Probabilistic Transition System (see Section 14.3.2)
$\mathbb{R}$	set of real numbers
$\mathscr{T}$	set of HOFL types (see Definition 7.2)
Tf	set of total functions from $\mathbb{N}$ to $\mathbb{N}_\perp$ (see Example 5.14)
Var	set of HOFL variables (see Chapter 7)
$\mathbb{Z}$	set of integers

DRAFT

Part III
HOFL: a higher-order functional language

DRAFT

This part focuses on models for sequential computations that are associated to HOFL, a higher-order declarative language that follows the functional style. Chapter 7 presents the syntax, typing and operational semantics of HOFL, while Chapter 9 defines its denotational semantics. The two are related in Chapter 10. Chapter 8 extends the theory presented in Chapter 5 to allow the definition of more complex domains, as needed by the type-constructors available in HOFL.

DRAFT

Chapter 9
Denotational Semantics of HOFL

Work out what you want to say before you decide how you want to say it. (Christopher Strachey's first law of logical design)

Abstract In this chapter we exploit the domain theory from Chapter 8 to define the (lazy) denotational semantics of HOFL. For each type τ we introduce a corresponding domain $(V_\tau)_\perp$ which is defined inductively over the structure of τ and such that we can assign an element of the domain $(V_\tau)_\perp$ to each (closed and typable) term t with type τ. Moreover, we introduce the notion of environment, which assigns meanings to variables, and that can be exploited to define the denotational semantics of (typable) terms with variables. Interestingly, all constructions we use are continuous, so that we are able to assign meaning also to any (typable) term that is recursively defined. We conclude the chapter by showing some important properties of the denotational semantics; in particular, that it is compositional.

DRAFT

9.1 HOFL Semantic Domains

In order to specify the denotational semantics of a programming language, we have to define, by structural recursion, an interpretation function from each syntactic domain to a semantic domain. In IMP there are three syntactic domains, *Aexp* for arithmetic expressions, *Bexp* for boolean expressions and *Com* for commands. Correspondingly, we have defined three semantics domains and three interpretation functions ($\mathscr{A}[\![\cdot]\!]$, $\mathscr{B}[\![\cdot]\!]$ and $\mathscr{C}[\![\cdot]\!]$). HOFL has a sole syntactic domain (i.e., the set of well-formed terms t) and thus we have only one interpretation function, written $[\![\cdot]\!]$. However, since HOFL terms are typed, the interpretation function is parametric w.r.t. the type τ of t and we have one semantic domain V_τ for each type τ. Actually, we distinguish between V_τ, where we find the meanings of the terms of type τ with canonical forms, and $(V_\tau)_\perp$, where the additional element $\perp_{(V_\tau)_\perp}$ assigns a meaning to all the terms of type τ without a canonical form. Moreover, we will need to handle terms with free variables, as, e.g., when defining the denotational semantics of $\lambda x.\, t$ in terms of the denotational semantics of t (with x possibly in $\mathrm{fv}(t)$). This was not the case for the operational semantics of HOFL, where only closed terms are considered. As terms may contain free variables, we pass to the interpretation function an *environment*

$$\rho \in Env \stackrel{\text{def}}{=} Var \to \bigcup_{\tau} (V_\tau)_\perp$$

which assigns meaning to variables. For consistency reasons, any environment ρ that we consider must satisfy the condition $\rho(x) \in (V_\tau)_\perp$ whenever $x : \tau$. Thus, we have

$$[\![t : \tau]\!] : Env \to (V_\tau)_\perp.$$

The actual semantic domains V_τ and $(V_\tau)_\perp$ are defined by structural recursion on the syntax of types:

$$\begin{array}{rcl@{\qquad}rcl} V_{int} & \stackrel{\text{def}}{=} & \mathbb{Z} & (V_{int})_\perp & \stackrel{\text{def}}{=} & \mathbb{Z}_\perp \\ V_{\tau_1 * \tau_2} & \stackrel{\text{def}}{=} & (V_{\tau_1})_\perp \times (V_{\tau_2})_\perp & (V_{\tau_1 * \tau_2})_\perp & \stackrel{\text{def}}{=} & ((V_{\tau_1})_\perp \times (V_{\tau_2})_\perp)_\perp \\ V_{\tau_1 \to \tau_2} & \stackrel{\text{def}}{=} & [(V_{\tau_1})_\perp \to (V_{\tau_2})_\perp] & (V_{\tau_1 \to \tau_2})_\perp & \stackrel{\text{def}}{=} & [(V_{\tau_1})_\perp \to (V_{\tau_2})_\perp]_\perp \end{array}$$

Notice that the recursive definition above takes advantage of the domain constructors we have defined in Chapter 8. While the lifiting $\mathbb{Z}_\perp$ of the integer numbers $\mathbb{Z}$ is strictly necessary, liftings on cartesian pairs and on continuous functions are actually optional, since cartesian products and functional domains are already $CPO_\perp$. We will discuss the motivation of our choice by the end of Chapter 10.

9.2 HOFL Interpretation Function

Now we are ready to define the interpretation function, by structural recursion. We briefly comment on each definition and show that the clauses of the structural recursion are typed correctly.

9.2.1 Constants

We define the meaning of a constant as the obvious value on the lifted domain:

$$[\![n]\!]\,\rho \stackrel{\text{def}}{=} \lfloor n \rfloor$$

At the level of types we have:

$$\underbrace{[\![\underbrace{n}_{int}]\!]\rho}_{(V_{int})_\perp = \mathbb{Z}_\perp} = \underbrace{\lfloor \underbrace{n}_{\mathbb{Z}} \rfloor}_{\mathbb{Z}_\perp}$$

DRAFT

9.2.2 Variables

The meaning of a variable is defined by its value in the given environment ρ:

$$[\![x]\!]\,\rho \stackrel{\text{def}}{=} \rho(x)$$

It is obvious that the typing is respected (under the assumption that $\rho(x) \in (V_\tau)_\perp$ whenever $x : \tau$):

$$\underbrace{[\![\underbrace{x}_{\tau}]\!]\rho}_{(V_\tau)_\perp} = \underbrace{\rho(\underbrace{x}_{\tau})}_{(V_\tau)_\perp}$$

9.2.3 Arithmetic Operators

We give the generic semantics of a binary operator $\mathrm{op} \in \{+,-,\times\}$ as:

$$[\![t_0 \ \mathrm{op}\ t_1]\!]\,\rho = [\![t_0]\!]\,\rho\ \underline{\mathrm{op}}_\perp\ [\![t_1]\!]\,\rho$$

where for any operator $\mathrm{op} \in \{+,-,\times\}$ in the syntax we have the corresponding function $\underline{\mathrm{op}} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ on the integers $\mathbb{Z}$ and also the binary function $\underline{\mathrm{op}}_\perp$ on $\mathbb{Z}_\perp$ defined as

$$\underline{\mathrm{op}}_\perp : (\mathbb{Z}_\perp \times \mathbb{Z}_\perp) \to \mathbb{Z}_\perp$$

$$x_1\ \underline{\mathrm{op}}_\perp\ x_2 = \begin{cases} \lfloor n_1\ \underline{op}\ n_2 \rfloor & \text{if } x_1 = \lfloor n_1 \rfloor \text{ and } x_2 = \lfloor n_2 \rfloor \text{ for some } n_1, n_2 \in \mathbb{Z} \\ \perp_{\mathbb{Z}_\perp} & \text{otherwise} \end{cases}$$

We remark that $\underline{\mathrm{op}}_\perp$ yields $\perp_{\mathbb{Z}_\perp}$ when at least one of the two arguments is $\perp_{\mathbb{Z}_\perp}$.

At the level of types, we have:

$$\underbrace{[\![(\underbrace{t_0}_{int}\ \mathrm{op}\ \underbrace{t_1}_{int})]\!]\rho}_{(V_{int})_\perp = \mathbb{Z}_\perp} = \underbrace{[\![\underbrace{t_0}_{int}]\!]\rho}_{(V_{int})_\perp}\ \underbrace{\underline{\mathrm{op}}_\perp}_{(\mathbb{Z}_\perp \times \mathbb{Z}_\perp) \to \mathbb{Z}_\perp}\ \underbrace{[\![\underbrace{t_1}_{int}]\!]\rho}_{(V_{int})_\perp}$$

9.2.4 Conditional

In order to define the semantics of the conditional expression, we exploit the conditional operator of the meta-language

$$Cond_\tau : \mathbb{Z}_\perp \times (V_\tau)_\perp \times (V_\tau)_\perp \to (V_\tau)_\perp$$

DRAFT

defined as:

$$Cond_\tau(v,d_0,d_1) \stackrel{\text{def}}{=} \begin{cases} d_0 & \text{if } v = \lfloor 0 \rfloor \\ d_1 & \text{if } \exists n \in \mathbb{Z}.\ v = \lfloor n \rfloor \wedge n \neq 0 \\ \bot_{(V_\tau)_\bot} & \text{if } v = \bot_{\mathbb{Z}_\bot} \end{cases}$$

Note that $Cond_\tau$ is parametric on the type τ. In the following, when τ can be inferred, we write just *Cond*. The conditional operator is *strict* on its first argument (i.e., it returns $\bot$ when the first argument is $\bot$) but not on the second and third arguments.

We can now define the denotational semantics of the conditional operator by letting:

$$[\![\textbf{if } t \textbf{ then } t_0 \textbf{ else } t_1]\!]\rho \stackrel{\text{def}}{=} Cond\,([\![t]\!]\rho, [\![t_0]\!]\rho, [\![t_1]\!]\rho)$$

At the level of types we have:

$$\underbrace{[\![\textbf{if } \underbrace{t_0}_{int} \textbf{ then } \underbrace{t_1}_{\tau} \textbf{ else } \underbrace{t_2}_{\tau}]\!]\rho}_{(V_\tau)_\bot} = \underbrace{\underbrace{Cond_\tau}_{\mathbb{Z}_\bot\times(V_\tau)_\bot\times(V_\tau)_\bot\to(V_\tau)_\bot}\ \underbrace{([\![t_0]\!]\rho, [\![t_1]\!]\rho, [\![t_2]\!]\rho)}_{(V_{int})_\bot\times(V_\tau)_\bot\times(V_\tau)_\bot}}_{(V_\tau)_\bot}$$

9.2.5 Pairing

For the pairing operator we simply let:

$$[\![(t_0,t_1)]\!]\rho \stackrel{\text{def}}{=} \lfloor ([\![t_0]\!]\rho, [\![t_1]\!]\rho) \rfloor$$

Note that, for $t_0 : \tau_0$ and $t_1 : \tau_1$, the pair $([\![t_0]\!]\rho, [\![t_1]\!]\rho)$ is in $(V_{\tau_0})_\bot \times (V_{\tau_1})_\bot$ and not in $((V_{\tau_0})_\bot \times (V_{\tau_1})_\bot)_\bot$, thus we apply the lifting. In fact, at the level of type consistency we have:

$$\underbrace{[\![\underbrace{(\underbrace{t_0}_{\tau_0}, \underbrace{t_1}_{\tau_1})}_{\tau_0*\tau_1}]\!]\rho}_{(V_{\tau_0*\tau_1})_\bot} = \underbrace{\lfloor ([\![\underbrace{t_0}_{\tau_0}]\!]\rho, [\![\underbrace{t_1}_{\tau_1}]\!]\rho) \rfloor}_{((V_{\tau_0})_\bot\times(V_{\tau_1})_\bot)_\bot}$$

9.2.6 Projections

We define the projections by using the lifted version of the projections π_1 and π_2 of the meta-language:

DRAFT

$$\begin{aligned}[\![\mathbf{fst}(t)]\!]\rho &\stackrel{\text{def}}{=} \mathbf{let}\ d \Leftarrow [\![t]\!]\rho.\ \pi_1\ d \\ &= \pi_1^*([\![t]\!]\rho) \\ [\![\mathbf{snd}(t)]\!]\rho &\stackrel{\text{def}}{=} \mathbf{let}\ d \Leftarrow [\![t]\!]\rho.\ \pi_2\ d \\ &= \pi_2^*([\![t]\!]\rho)\end{aligned}$$

The **let** operator (see Definition 8.10) allows to *de-lift* $[\![t]\!]\rho$ in order to apply projections π_1 and π_2. Instead, if $[\![t]\!]\rho = \bot$ the result is also $\bot$.

Again, we check that the type constraints are respected by the definition:

$$[\![\mathbf{fst}(\underbrace{\underbrace{t}_{\tau_0*\tau_1}}_{\tau_0})]\!]\rho = \mathbf{let}\ \underbrace{d}_{V_{\tau_0*\tau_1}} \Leftarrow \underbrace{[\![\underbrace{t}_{\tau_0*\tau_1}]\!]\rho}_{(V_{\tau_0*\tau_1})_\bot}.\ \underbrace{\underbrace{\pi_1}_{(V_{\tau_0})_\bot\times(V_{\tau_1})_\bot\to(V_{\tau_0})_\bot}\ \underbrace{d}_{V_{\tau_0*\tau_1}}}_{(V_{\tau_0})_\bot}$$

The case of $\mathbf{snd}(t : \tau_0 * \tau_1)$ is completely analogous and thus omitted.

9.2.7 Lambda Abstraction

For lambda-abstraction we use, of course, the lambda operator of the meta-language:

$$[\![\lambda x.\ t]\!]\rho \stackrel{\text{def}}{=} \left\lfloor \lambda d.\ [\![t]\!]\rho[^d/_x] \right\rfloor$$

where we bind x to d for evaluating t.

Note that, as in the case of pairing, we need to apply the lifting, because $\lambda d.\ [\![t]\!]\rho[^d/_x]$ is an element of $V_{\tau_0\to\tau_1} = [(V_{\tau_0})_\bot \to (V_{\tau_1})_\bot]$ and not of $(V_{\tau_0\to\tau_1})_\bot = [(V_{\tau_0})_\bot \to (V_{\tau_1})_\bot]_\bot$.

$$[\![\lambda\underbrace{\underbrace{x}_{\tau_0}.\ \underbrace{t}_{\tau_1}}_{\tau_0\to\tau_1}]\!]\rho = \lfloor\underbrace{\underbrace{\lambda\ \underbrace{d}_{(V_{\tau_0})_\bot}.\ \underbrace{[\![\underbrace{t}_{\tau_1}]\!]\rho[^d/_x]}_{(V_{\tau_1})_\bot}}_{[(V_{\tau_0})_\bot\to(V_{\tau_1})_\bot]}}_{[(V_{\tau_0})_\bot\to(V_{\tau_1})_\bot]_\bot}\rfloor$$

9.2.8 Function Application

Similarly to the case of projections, we apply the de-lifted version of the function to its argument:

DRAFT

$$\begin{aligned}[\![(t_1\ t_0)]\!]\rho &\stackrel{\text{def}}{=} \textbf{let}\ \varphi \Leftarrow [\![t_1]\!]\rho.\ \varphi([\![t_0]\!]\rho)\\ &= (\lambda\varphi.\ \varphi([\![t_0]\!]\rho))^*([\![t_1]\!]\rho)\end{aligned}$$

At the level of types, we have:

$$[\![(\underbrace{\underbrace{t_1}_{\tau_0\to\tau_1}\ \underbrace{t_0}_{\tau_0}}_{\tau_1})]\!]\rho = \underbrace{\textbf{let}\ \underbrace{\varphi}_{V_{\tau_0\to\tau_1}} \Leftarrow \underbrace{[\![t_1]\!]\rho}_{(V_{\tau_0\to\tau_1})_\perp}.\ \underbrace{\underbrace{\varphi}_{V_{\tau_0\to\tau_1}}\ \underbrace{([\![t_0]\!]\rho)}_{(V_{\tau_0})_\perp}}_{(V_{\tau_1})_\perp}}_{(V_{\tau_1})_\perp}$$

9.2.9 Recursion

For handling recursion we would like to find a solution (in the domain $(V_\tau)_\perp$, for $t:\tau$) to the recursive equation

$$[\![\textbf{rec}\ x.\ t]\!]\rho = [\![t]\!]\rho[{}^{[\![\textbf{rec}\ x.\ t]\!]\rho}/_x]$$

The least solution can be computed simply by applying the fix operator of the meta-language:

$$[\![\textbf{rec}\ x.\ t]\!]\rho \stackrel{\text{def}}{=} \text{fix}\ \lambda d.\ [\![t]\!]\rho[{}^d/_x]$$

Finally, we check that also this last definition is consistent with the typing:

$$[\![\textbf{rec}\ \underbrace{x}_{\tau}.\ \underbrace{t}_{\tau}]\!]\rho = \underbrace{\underbrace{\underbrace{\text{fix}}_{[[(V_\tau)_\perp\to(V_\tau)_\perp]\to(V_\tau)_\perp]}\ \underbrace{\lambda\ \underbrace{d}_{(V_\tau)_\perp}.\ \underbrace{[\![\underbrace{t}_{\tau}]\!]\rho[{}^d/_x]}_{(V_\tau)_\perp}}_{[(V_\tau)_\perp\to(V_\tau)_\perp]}}_{(V_\tau)_\perp}}_{(V_\tau)_\perp}$$

9.2.10 Examples

Example 9.1. Let us see some simple examples of evaluation of the denotational semantics. We consider three similar terms f,g,h such that f and h have the same denotational semantics while g has a different semantics because it requires a parameter x to be evaluated even if not used.

1. $f \stackrel{\text{def}}{=} \lambda x:int.\ 3$
2. $g \stackrel{\text{def}}{=} \lambda x:int.\ \textbf{if}\ x\ \textbf{then}\ 3\ \textbf{else}\ 3$
3. $h \stackrel{\text{def}}{=} \textbf{rec}\, y:int\to int.\ \lambda x:int.\ 3$

Note that $f,g,h:int\to int$. For the term f we have:

DRAFT

$$[\![f]\!]\rho = [\![\lambda x.\ 3]\!]\rho = \lfloor \lambda d.\ [\![3]\!]\rho[^{d}/_{x}] \rfloor = \lfloor \lambda d.\ \lfloor 3 \rfloor \rfloor$$

When considering g, instead:

$$\begin{aligned}
[\![g]\!]\rho &= [\![\lambda x.\ \textbf{if } x \textbf{ then } 3 \textbf{ else } 3]\!]\rho \\
&= \lfloor \lambda d.\ [\![\textbf{if } x \textbf{ then } 3 \textbf{ else } 3]\!]\rho[^{d}/_{x}] \rfloor \\
&= \lfloor \lambda d.\ \mathit{Cond}(d, \lfloor 3 \rfloor, \lfloor 3 \rfloor) \rfloor \\
&= \lfloor \lambda d.\ \textbf{let } x \Leftarrow d.\ \lfloor 3 \rfloor \rfloor
\end{aligned}$$

where the last equality follows from the fact that both expressions $\mathit{Cond}(d, \lfloor 3 \rfloor, \lfloor 3 \rfloor)$ and **let** $x \Leftarrow d.\ \lfloor 3 \rfloor$ evaluate to $\bot_{\mathbb{Z}_\bot}$ when $d = \bot_{\mathbb{Z}_\bot}$ and to $\lfloor 3 \rfloor$ if d is a lifted value. Thus we can conclude that $[\![f]\!]\rho \neq [\![g]\!]\rho$.

Finally, for h we get:

$$\begin{aligned}
[\![h]\!]\rho &= [\![\textbf{rec } y.\ \lambda x.\ 3]\!]\rho \\
&= \text{fix } \lambda d_y.\ [\![\lambda x.\ 3]\!]\rho[^{d_y}/_{y}] \\
&= \text{fix } \lambda d_y.\ \lfloor \lambda d_x.\ [\![3]\!]\rho[^{d_y}/_{y}, ^{d_x}/_{x}] \rfloor \\
&= \text{fix } \lambda d_y.\ \lfloor \lambda d_x.\ \lfloor 3 \rfloor \rfloor
\end{aligned}$$

Let $\Gamma_h = \lambda d_y.\ \lfloor \lambda d_x.\ \lfloor 3 \rfloor \rfloor$. We can compute the fixpoint by exploiting the fixpoint theorem to compute successive approximations:

$$\begin{aligned}
d_0 &= \Gamma_h^0(\bot_{[\mathbb{Z}_\bot \to \mathbb{Z}_\bot]_\bot}) = \bot_{[\mathbb{Z}_\bot \to \mathbb{Z}_\bot]_\bot} \\
d_1 &= \Gamma_h(d_0) = (\lambda d_y.\ \lfloor \lambda d_x.\ \lfloor 3 \rfloor \rfloor)\bot = \lfloor \lambda d_x.\ \lfloor 3 \rfloor \rfloor \\
d_2 &= \Gamma_h(d_1) = (\lambda d_y.\ \lfloor \lambda d_x.\ \lfloor 3 \rfloor \rfloor)\lfloor \lambda d_x.\ \lfloor 3 \rfloor \rfloor = \lfloor \lambda d_x.\ \lfloor 3 \rfloor \rfloor = d_1
\end{aligned}$$

Since $d_2 = d_1$ we have reached the fixpoint and thus

$$[\![h]\!]\rho = \lfloor \lambda d_x.\ \lfloor 3 \rfloor \rfloor = [\![f]\!]\rho.$$

Note that we could have avoided the calculation of d_2, because d_1 is already a maximal element in $[\mathbb{Z}_\bot \to \mathbb{Z}_\bot]_\bot$ and therefore it must be $\Gamma_h(d_1) = d_1$.

9.3 Continuity of Meta-language's Functions

In order to show that the semantics is always well defined we have to show that all the functions we employ in the definition are continuous, so that the fixpoint theory is applicable.

Theorem 9.1. *The following functions are monotone and continuous:*

1. $\underline{\text{op}}_\bot : (\mathbb{Z}_\bot \times \mathbb{Z}_\bot) \to \mathbb{Z}_\bot$;
2. $\mathit{Cond}_\tau : \mathbb{Z}_\bot \times (V_\tau)_\bot \times (V_\tau)_\bot \to (V_\tau)_\bot$;

DRAFT

3. $(\_,\_) : (V_{\tau_0})_\perp \times (V_{\tau_1})_\perp \to V_{\tau_0 * \tau_1}$;
4. $\pi_1 : V_{\tau_0 * \tau_1} \to (V_{\tau_0})_\perp$;
5. $\pi_2 : V_{\tau_0 * \tau_1} \to (V_{\tau_1})_\perp$;
6. **let**
7. apply
8. $\mathrm{fix} : [[(V_\tau)_\perp \to (V_\tau)_\perp] \to (V_\tau)_\perp]$.

Proof. Monotonocity is obvious in most cases. We focus on the continuity of the various functions

1. Since $\underline{\mathrm{op}}_\perp$ is monotone over a domain with only finite chains then it is also continuous.
2. By using the Theorem 8.7, we can prove the continuity of *Cond* on each parameter separately.
   Let us show the continuity on the first parameter. Since chains in $\mathbb{Z}_\perp$ are finite, it is enough to prove monotonicity. We fix $d_1, d_2 \in (V_\tau)_\perp$ and we prove the monotonicity of $\lambda x.\, Cond_\tau(x, d_1, d_2) : \mathbb{Z}_\perp \to (V_\tau)_\perp$. Let $n, m \in \mathbb{Z}$.

   - the cases $\perp_{\mathbb{Z}_\perp} \sqsubseteq_{\mathbb{Z}_\perp} \perp_{\mathbb{Z}_\perp}$ or $\lfloor n \rfloor \sqsubseteq_{\mathbb{Z}_\perp} \lfloor n \rfloor$ are trivial;
   - for the case $\perp_{\mathbb{Z}_\perp} \sqsubseteq_{\mathbb{Z}_\perp} \lfloor n \rfloor$ then obviously

     $$Cond_\tau(\perp_{\mathbb{Z}_\perp}, d_1, d_2) = \perp_{(V_\tau)_\perp} \sqsubseteq_{(V_\tau)_\perp} Cond_\tau(\lfloor n \rfloor, d_1, d_2)$$

     because $\perp_{(V_\tau)_\perp}$ is the bottom element of $(V_\tau)_\perp$.
   - for the case $\lfloor n \rfloor \sqsubseteq_{\mathbb{Z}_\perp} \lfloor m \rfloor$, since $\mathbb{Z}_\perp$ is a flat domain we have $n = m$ and trivially $Cond_\tau(\lfloor n \rfloor, d_1, d_2) \sqsubseteq_{(V_\tau)_\perp} Cond_\tau(\lfloor m \rfloor, d_1, d_2)$

   Now let us show the continuity on the second parameter, namely we fix $v \in \mathbb{Z}_\perp$ and $d \in (V_\tau)_\perp$ and for any chain $\{d_i\}_{i \in \mathbb{N}}$ in $(V_\tau)_\perp$ we prove that

   $$Cond_\tau\left(v, \bigsqcup_{i \in \mathbb{N}} d_i, d\right) = \bigsqcup_{i \in \mathbb{N}} Cond_\tau(v, d_i, d)$$

   - if $v = \perp_{\mathbb{Z}_\perp}$, then

     $$Cond_\tau\left(\perp_{\mathbb{Z}_\perp}, \bigsqcup_{i \in \mathbb{N}} d_i, d\right) = \perp_{\mathbb{Z}_\perp} = \bigsqcup_{i \in \mathbb{N}} \perp_{\mathbb{Z}_\perp} = \bigsqcup_{i \in \mathbb{N}} Cond_\tau(\perp_{\mathbb{Z}_\perp}, d_i, d)$$

   - if $v = \lfloor 0 \rfloor$, then $\lambda x.\, Cond_\tau(\lfloor 0 \rfloor, x, d)$ is the identity function $\lambda x.\, x$ and we have

     $$Cond_\tau\left(\lfloor 0 \rfloor, \bigsqcup_{i \in \mathbb{N}} d_i, d\right) = \bigsqcup_{i \in \mathbb{N}} d_i = \bigsqcup_{i \in \mathbb{N}} Cond_\tau(\lfloor 0 \rfloor, d_i, d)$$

   - if $v = \lfloor n \rfloor$ with $n \neq 0$, then $\lambda x.\, Cond_\tau(\lfloor n \rfloor, x, d)$ is the constant function $\lambda x.\, d$ and we have

DRAFT

$$Cond_\tau\left(\lfloor n\rfloor, \bigsqcup_{i\in\mathbb{N}} d_i, d\right) = d = \bigsqcup_{i\in\mathbb{N}} d = \bigsqcup_{i\in\mathbb{N}} Cond_\tau(\lfloor n\rfloor, d_i, d)$$

In all cases $Cond_\tau$ is continuous.
Continuity on the third parameter is analogous.

3. For pairing $(\_,\_)$ we can use again the Theorem 8.7, which allows to show separately the continuity on each parameter. If we fix the first element we have

$$\left(d, \bigsqcup_{i\in\mathbb{N}} d_i\right) = \left(\bigsqcup_{i\in\mathbb{N}} d, \bigsqcup_{i\in\mathbb{N}} d_i\right) = \bigsqcup_{i\in\mathbb{N}} (d, d_i)$$

by definition of lub of a chain of pairs (see Theorem 8.1). The same holds for the second parameter.
4. Projections π_1 and π_2 are continuous by Theorem 8.2.
5. The **let** function is continuous since $(\cdot)^*$ is continuous by Theorem 8.4.
6. apply is continuous by Theorem 8.8
7. fix is continuous by Theorem 8.10. □

In the previous theorem we have not mentioned the continuity proofs for lambda abstraction and recursion. The next theorem fills these gaps.

Theorem 9.2. *Let $t : \tau$ be a well typed term of HOFL; then the following holds:*

1. $(\lambda d.\ [\![t]\!]\rho[^d/_x])$ *is a continuous function.*
2. fix $\lambda d.\ [\![t]\!]\rho[^d/_x]$ *is a continuous function.*

Proof. Let us prove the two properties:

1. We prove the stronger property that, for any $n \in \mathbb{N}$:

$$\lambda(d_1,...,d_n).\ [\![t]\!]\rho[^{d_1}/_{x_1}, \cdots, ^{d_n}/_{x_n}]$$

is a continuous function. The proof is by structural induction on t. Below, for brevity, we write $\widetilde{d}$ instead of $d_1,...,d_n$ and ρ' instead of $\rho[^{d_1}/_{x_1}, \cdots, ^{d_n}/_{x_n}]$:

$t = y$: Then $\lambda\widetilde{d}.\ [\![y]\!]\rho'$ is either a projection function (if $y = x_i$ for some $i \in [1,n]$) or the constant function $\lambda\widetilde{d}.\ \rho(y)$ (if $y \notin \{x_1,...,x_n\}$), which are continuous.

$t = t_1\ \mathrm{op}\ t_2$: By inductive hypothesis $f_1 \stackrel{\text{def}}{=} \lambda\widetilde{d}.\ [\![t_1]\!]\rho'$ and $f_2 \stackrel{\text{def}}{=} \lambda\widetilde{d}.\ [\![t_2]\!]\rho'$ are continuous. Then $f \stackrel{\text{def}}{=} \lambda\widetilde{d}.\ ((f_1\ \widetilde{d}),(f_2\ \widetilde{d}))$ is continuous, and

$$\begin{aligned}\lambda\widetilde{d}.\ [\![t_1\ \mathrm{op}\ t_2]\!]\rho' &= \lambda\widetilde{d}.\ ([\![t_1]\!]\rho'\ \underline{\mathrm{op}}_\perp\ [\![t_2]\!]\rho')\\ &= \lambda\widetilde{d}.\ (f_1\ \widetilde{d})\ \underline{\mathrm{op}}_\perp\ (f_2\ \widetilde{d})\\ &= \underline{\mathrm{op}}_\perp \circ f\end{aligned}$$

is continuous because $\underline{\mathrm{op}}_\perp$ is continuous and the composition of continuous functions yields a continuous function by Theorem 8.5.

DRAFT

$t = \lambda y.\, t'$: By induction hypothesis we can assume that $\lambda(\widetilde{d},d).[\![t']\!]\rho'[^d/_y]$ is continuous. Then $\mathrm{curry}(\lambda(\widetilde{d},d).[\![t']\!]\rho'[^d/_y])$ is continuous since curry is continuous, and we conclude by noting that

$$\begin{aligned}\mathrm{curry}(\lambda(\widetilde{d},d).[\![t']\!]\rho'[^d/_y]) &= \lambda\widetilde{d}.\,\lambda d.\,[\![t']\!]\rho'[^d/_y]\\ &= \lambda\widetilde{d}.[\![\lambda y.\,t']\!]\rho'.\end{aligned}$$

We leave the remaining cases as an exercise.

2. To prove the second proposition we note that

$$\mathrm{fix}\ \lambda d.[\![t]\!]\rho[^d/_x]$$

is the application of a continuous function (i.e., the function fix, by Theorem 8.10) to a continuous argument (i.e., $\lambda d.[\![t]\!]\rho[^d/_x]$, continuous by the first part of this theorem) so it is continuous by Theorem 8.8. □

We conclude this section by recalling that the definition of denotational semantics is consistent with the typing.

Theorem 9.3 (Type Consistency). *If $t : \tau$ then $[\![t]\!]\rho \in (V_\tau)_\perp$.*

Proof. The proof is by structural induction on t and it has been outlined when giving the structurally recursive definition of the denotational semantics (where we have also relied on the previous continuity theorems). □

9.4 Substitution Lemma and Other Properties

We conclude this chapter by stating some useful theorems. The most important is the *Substitution Lemma* which states that the substitution operator commutes with the interpretation function.

Theorem 9.4 (Substitution Lemma). *Let $x,t : \tau$ and $t' : \tau'$. We have*

$$[\![t'[^t/_x]]\!]\rho = [\![t']\!]\rho[^{[\![t]\!]\rho}/_x]$$

Proof. By Theorem 7.1 we know that $t'[^t/_x] : \tau'$. The proof is by structural induction on t' and left as an exercise (see Problem 9.13). □

In words, replacing a variable x with a term t in a term t' returns a term $t'[^t/_x]$ whose denotational semantics $[\![t'[^t/_x]]\!]\rho = [\![t']\!]\rho[^{[\![t]\!]\rho}/_x]$ depends only on the denotational semantics $[\![t]\!]\rho$ of t.

Remark 9.1 (Compositionality). The substitution lemma is an important result, as it implies the compositionality of denotational semantics, namely for all terms t_1, t_2 and environment ρ we have:

$$[\![t_1]\!]\rho = [\![t_2]\!]\rho \quad \Rightarrow \quad [\![t[^{t_1}/_x]]\!]\rho = [\![t[^{t_2}/_x]]\!]\rho$$

DRAFT

Theorem 9.5. *Let t be a well-defined term of HOFL. Let $\rho, \rho' \in Env$ such that $\forall\, x \in \mathrm{fv}(t).\ \rho(x) = \rho'(x)$ then:*

$$[\![t]\!]\,\rho = [\![t]\!]\,\rho'$$

Proof. The proof is by structural induction on t and left as an exercise (see Problem 9.16). □

Theorem 9.6. *Let $c \in C_\tau$ be a closed term in canonical form of type τ. Then we have:*

$$\forall\, \rho \in Env.\ [\![c]\!]\,\rho \neq \bot_{(V_\tau)_\bot}$$

Proof. Immediate, by inspection of the clauses for terms in canonical forms. □

Problems

9.1. Consider the HOFL term:

$$t \stackrel{\text{def}}{=} \textbf{rec}\ f.\ \lambda x.\ \textbf{if}\ x\ \textbf{then}\ 0\ \textbf{else}\ (f(x) \times f(x))$$

Derive the type, the canonical form and the denotational semantics of t.

9.2. Consider the HOFL term:

$$t \stackrel{\text{def}}{=} \textbf{rec}\ f.\ \lambda x.\ \lambda y.\ \textbf{if}\ x \times y\ \textbf{then}\ x\ \textbf{else}\ (fx)((fx)y)$$

Derive the type, the canonical form and the denotational semantics of t.

9.3. Consider the HOFL term:

$$t \stackrel{\text{def}}{=} \textbf{fst}(\ (\lambda x.\ x)\ (\ 1\ ,\ ((\textbf{rec}\, f.\ \lambda y.\ (f\ y))\ 2)\ )\ ).$$

Derive the type, the canonical form and the denotational semantics of t.

9.4. Consider the HOFL term

$$t \stackrel{\text{def}}{=} \textbf{rec}\ f.\ \lambda x.\ \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ (g\ (f\ (x-1)))$$

1. Derive the type of t and the denotational semantics of $[\![t]\!]\,\rho$ by assuming that $\rho g = \lfloor h \rfloor$ for some suitable h.
2. Compute the canonical form of the term $(((\lambda g.\ t)\ \lambda x.\ x)\ 1)$. Would it be possible to compute the canonical form of t?

9.5. Let us consider the following recursive definition:

$$f(x) \stackrel{\text{def}}{=} \textbf{if}\ x = 0\ \textbf{then}\ 1\ \textbf{else}\ 2 \times f(x-1).$$

DRAFT

1. Define a well-formed, closed HOFL term t that corresponds to the above definition and determine its type.
2. Compute its denotational semantics $[\![t]\!]\rho$ and prove that

$$n \geq 0 \quad \Rightarrow \quad \textbf{let } \varphi \Leftarrow [\![t]\!]\rho.\ \varphi\lfloor n \rfloor = \lfloor 2^n \rfloor.$$

Hint: Prove that the n-th fixpoint approximation is

$$d_n \ = \ \lfloor\ \lambda d.\ Cond(\lfloor 0 \rfloor \underline{\leq}_{\perp} d \underline{\leq}_{\perp} \lfloor n \rfloor\ ,\ \lfloor 2^d \rfloor\ ,\ \perp)\ \rfloor.$$

9.6. Let us consider the following recursive definition:

$$f(x) \stackrel{\text{def}}{=} \textbf{if } x = 0 \textbf{ then } 0 \textbf{ else } f(f(x-1))$$

1. Define a well-formed, closed HOFL term t that corresponds to the above definition and determine its type, its canonical form and its denotational semantics.
2. Define the set of fixpoints that satisfy the recursive definition.

9.7. Consider the HOFL term

$$t \stackrel{\text{def}}{=} \textbf{rec } f.\ \lambda x.\ \textbf{if } x \textbf{ then } 0 \textbf{ else } f\ (x-x)$$

1. Determine the type of t and its denotational semantics $[\![t]\!]\rho = \text{fix } \Gamma$.
2. Is fix Γ the unique fixpoint of Γ?
   Hint: Consider the elements greater than fix Γ in the order and check if they are fixpoints for Γ.

9.8. Consider the Fibonacci sequence already found in Problem 4.14 and the corresponding term t from Problem 7.8:

$$F(0) \stackrel{\text{def}}{=} 1 \qquad F(1) \stackrel{\text{def}}{=} 1 \qquad F(n+2) \stackrel{\text{def}}{=} F(n+1) + F(n)s.$$

where $n \in \mathbb{N}$.

1. Compute the suitable transformation Γ such that $[\![t]\!]\rho = \text{fix } \Gamma$.
2. Prove that the denotational semantics $[\![t]\!]\rho$ satisfies the above equations, to conclude that the given implementation of Fibonacci numbers is correct.
   Hint: Compute $[\![(t\ 0)]\!]\rho$, $[\![(t\ 1)]\!]\rho$ and $[\![(t\ n+2)]\!]\rho$ exploiting the equality $[\![t]\!] = \Gamma\,[\![t]\!]$.

9.9. Assuming that t_1 has type τ_1, let us consider the term $t_2 \stackrel{\text{def}}{=} \lambda x.\ (t_1\ x)$.

1. Do both terms have the same type?
2. Do both terms have the same lazy denotational semantics?

9.10. Let us consider the terms

$$t_1 \stackrel{\text{def}}{=} \lambda x.\ \textbf{rec } y.\ y+1$$
$$t_2 \stackrel{\text{def}}{=} \textbf{rec } y.\ \lambda x.\ (y\ x)+2$$

DRAFT

1. Do both terms have the same type?
2. Do both terms have the same lazy denotational semantics?

9.11. Given a monotone function $f : \mathbb{Z}_\perp \to \mathbb{Z}_\perp$, prove that $f\perp_{\mathbb{Z}_\perp} = f(f\perp_{\mathbb{Z}_\perp})$. Then, let $t : \mathit{int} \to \mathit{int}$ be a closed term of HOFL and consider the term

$$t_1 \stackrel{\text{def}}{=} \mathbf{rec}\ f.\ \lambda x.\ (t\ (f\ x))$$

1. Determine the most general type of t_1.
2. Exploit the above result to prove that $[\![t_1]\!]\rho = [\![t_2]\!]\rho$, where

$$t_2 \stackrel{\text{def}}{=} \mathbf{rec}\ f.\ \lambda x.\ (t\ \mathbf{rec}\ y.\ y)$$

9.12. Let us extend the syntax of (lazy) HOFL by adding the construct for sequential composition $t_1;t_2$ that, informally, represents the function obtained by applying the function t_1 to the argument and then the function t_2 to the result. Define, for the new construct ;:

1. the typing rule;
2. the (big-step) operational semantics;
3. the denotational semantics.

Then prove that for every closed term t, both terms $(t_1;t_2\ t)$ and $(t_2\ (t_1\ t))$ have the same type and are equivalent according to the denotational semantics.

9.13. Complete the proof of the Substitution Lemma (Theorem 9.4).

9.14. Let t_1, t_2 be well-formed HOFL terms and ρ an environment.

1. Prove that

$$[\![t_1]\!]\rho\ =\ [\![t_2]\!]\rho \quad \Rightarrow \quad [\![(t_1\ x)]\!]\rho\ =\ [\![(t_2\ x)]\!]\rho \tag{9.1}$$

2. Prove that the reversed implication is generally not valid by giving a counterexample. Then, find the conditions under which also the reversed implication holds.
3. Exploit the Substitution Lemma (Theorem 9.4) to prove that for all t and $x \notin \mathrm{fv}(t_1) \cup \mathrm{fv}(t_2)$:

$$[\![t_1]\!]\rho\ =\ [\![t_2]\!]\rho \quad \Rightarrow \quad [\![t[t_1/x]]\!]\rho\ =\ [\![t[t_2/x]]\!]\rho \tag{9.2}$$

4. Observe that the implication 9.1 is just a special case of the latter equality 9.2 and explain how.

9.15. Is it possible to modify the denotational semantics of HOFL assigning to the construct

$$\mathbf{if}\ t\ \mathbf{then}\ t_0\ \mathbf{else}\ t_1$$

- the semantics of t_1 if the semantics of t is $\perp_{N_\perp}$, and

DRAFT

- the semantics of t_0 otherwise? (If not, why?)

9.16. Complete the proof of Theorem 9.5.

DRAFT