Mendler Induction
and Classical Logic

Marco António Ferreira Devesas Campos
Corpus Christi College

December 2015

A dissertation submitted for the degree of
Doctor of Philosophy
Declaration

All the work herein is the original contribution of the author unless stated otherwise. No part of it has been submitted for the obtainment of a degree at the University of Cambridge, or elsewhere. This dissertation does not exceed the regulation 60,000 words, including tables and footnotes.
Aos meus Pais
Mendler Induction and Classical Logic
Marco António Ferreira Devesas Campos

Abstract

This thesis lies at the intersection of two fields: Computer Science and Logic. It follows in the tradition of studies on the Curry-Howard correspondence between types and propositions, and between programs and proofs.

We commence from the Dual Calculus, a language which exhibits interesting computational behaviour by including control effects—the ability for a piece of code to alter its running context. Such manipulations are known to correspond logically to laws such as Peirce’s, the excluded middle, or double negation elimination. For this we dub this language Classical. Our study of the Dual Calculus is guided by a comparative effort between the programming language and Gentzen’s related seminal system, the Sequent Calculus; and we refine the (oft presumed) connection between the two. Our discussion is quite general, spanning both propositional—including negation—and quantified types, and, less familiarly, subtractive types.

Next we turn to the reduction rules of the calculus and show that they satisfy several properties: self-duality, substitutivity, subject preservation and, strong normalization—this despite them being non-deterministic and inherently non-confluent. We prove strong normalization by recourse to a very powerful and general realizability argument. Further, we synthesize from our model a lattice structure that is not present in other realizability proofs of strong normalization.

The second half of the thesis focuses on Mendler induction—first in a functional setting, and then within the Dual Calculus. The functional setting serves to motivate and explicate this lesser known form of induction that is, nonetheless, more general than the usual notion—which it subsumes—in that it enables well-founded induction on types with negative occurrences of the induction variable. The definition of the necessary extension of the Dual Calculus follows. Subtractive types are paramount to this definition.

The system thus created is very general, and can embed several other calculi. We show in particular two: the original Mendler inductive (functional) system and a (Classical) Dual Calculus with inductive types. But despite this generality, we still retain the nice reduction behaviour of the of the second-order system—even if this is shown only classically.
Acknowledgements

My first thank you goes to Professors Luís Soares Barbosa and João Nuno Oliveira at Minho University—without their teachings, collaboration, and support, the Cambridge adventure that culminated in this thesis would have never even begun.

I would like to thank Marcelo Fiore for acting as my supervisor and introducing me to the wonderful world of Mendler induction; and for teaching me far more than I ever thought my natural skills would allow. All the failings of this work—including but not limited to its excessive flamboyance—are my exclusive doing. I’d like to thank Steffen Lösch, and Ohad Kammar for assorted, research-stimulating conversations. Professors Andy Pitts and Tim Griffin examined my first and second year reports and made incisive comments that shaped the contents and structure of this thesis immensely—all the while being very supportive of my research.

I’d like to acknowledge the financial support of the United Kingdom’s Engineering and Physical Sciences Research Council (grant number EP/J500380/1), and the Computer Laboratory—especially Lise Gough for handling all the bureaucracies for me. Also, through Megan Sammons’ incessant efforts organising supervisions, I was able to gain invaluable experience in teaching, and earn much needed additional funds. I would like to thank Corpus Christi College for their financial patience.

One a different note, a heartfelt thank you goes out to everyone who has been at Corpus Christi College over the past five years—especially the members of its Middle Combination Room, but also its staff and even the pesky undergraduates—for surrounding me with a vibrant social environment where I could find some respite from research. I started writing a list of people who I’d like to thank individually, but had to stop for fear it would get me over the word limit! However, I do need to give special mention to Alex Carr, Gyuseong Cho, Daisy Mechelmans, Kai Ruggeri, Marcin Ziarko, and Ben and Jo Watson for going above and beyond their friendship duties and providing me with roof during the final bits of writing.

Last, but certainly not least, this thesis is dedicated to my parents—for their unwavering support, for their enthusiastic encouragement, for their unconditional love. Though they will understand very little of it, this work is as much theirs as it is mine.
# Contents

1 Introduction ........................................... 11
   1.1 The Curry-Howard Isomorphism ...................... 11
   1.2 Classical Logic and Control .......................... 13
   1.3 Mendler Induction ................................... 14
   1.4 Thesis ............................................. 15

2 The Dual Calculus ..................................... 17
   2.1 Gentzen’s LK ....................................... 17
   2.2 Wadler's Dual Calculus .............................. 26
      2.2.1 Syntax ........................................ 26
      2.2.2 Typing ........................................ 28
   2.3 Proving and Programming with DC .................... 32
   2.4 Reduction ........................................ 37
   2.5 Extensions of the Dual Calculus ................. 39
   2.6 Curry-Howard Correspondence of LK and the Dual Calculus 41

3 Reduction Properties of the Dual Calculus ............. 47
   3.1 Parallel Reduction .................................. 47
   3.2 Duality ........................................... 48
   3.3 Substitutivity ..................................... 52
   3.4 Preservation ....................................... 57
   3.5 Strong Normalization .............................. 63
      3.5.1 Sets and Saturation .......................... 64
      3.5.2 Orthogonal Pairs ............................ 70
      3.5.3 Orthogonal Actions .......................... 74
      3.5.4 Structural Completion ....................... 77
      3.5.5 Interpretation of Types ..................... 81
      3.5.6 Strong Normalization ...................... 87

4 Mendler Induction ..................................... 97
   4.1 Mendler’s Induction Operator ..................... 98
   4.2 Negative Typeschemes and Mendler Induction .... 105
   4.3 Mendler and Ordinary Induction .................... 107
## List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The identity and structural rules of Gentzen’s LK (Gentzen, 1964)</td>
<td>19</td>
</tr>
<tr>
<td>2.2</td>
<td>The operational rules of Gentzen’s LK (Gentzen, 1964)</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>The duality of formulas in LK</td>
<td>24</td>
</tr>
<tr>
<td>2.4</td>
<td>The types of the Dual Calculus</td>
<td>26</td>
</tr>
<tr>
<td>2.5</td>
<td>The syntax of the Dual Calculus</td>
<td>27</td>
</tr>
<tr>
<td>2.6</td>
<td>Structural typing rules of the Dual Calculus</td>
<td>30</td>
</tr>
<tr>
<td>2.7</td>
<td>Operational typing rules for the Dual Calculus</td>
<td>31</td>
</tr>
<tr>
<td>2.8</td>
<td>Typing rules for abstractions in the Dual Calculus</td>
<td>31</td>
</tr>
<tr>
<td>2.9</td>
<td>Duality of DC types and contexts</td>
<td>36</td>
</tr>
<tr>
<td>2.10</td>
<td>Duality for the syntax of DC</td>
<td>36</td>
</tr>
<tr>
<td>2.11</td>
<td>Operational β-reduction rules of DC</td>
<td>38</td>
</tr>
<tr>
<td>2.12</td>
<td>The η-equalities of DC</td>
<td>38</td>
</tr>
<tr>
<td>2.13</td>
<td>Abstraction β-reduction rules</td>
<td>38</td>
</tr>
<tr>
<td>2.14</td>
<td>The co-abstraction first subsystem of DC</td>
<td>39</td>
</tr>
<tr>
<td>2.15</td>
<td>The abstraction first subsystem of DC</td>
<td>40</td>
</tr>
<tr>
<td>2.16</td>
<td>Terms and typings of the implicative fragment of DC</td>
<td>41</td>
</tr>
<tr>
<td>2.17</td>
<td>Reduction rules for the implicative fragment of DC</td>
<td>41</td>
</tr>
<tr>
<td>2.18</td>
<td>Second-order terms and typings</td>
<td>42</td>
</tr>
<tr>
<td>2.19</td>
<td>Reduction for second-order cuts</td>
<td>42</td>
</tr>
<tr>
<td>2.20</td>
<td>Summary of the typing and reduction rules of the second-order Dual Calculus (I of II)</td>
<td>44</td>
</tr>
<tr>
<td>2.21</td>
<td>Summary of the typing and reduction rules of the second-order Dual Calculus (II of II)</td>
<td>45</td>
</tr>
<tr>
<td>3.1</td>
<td>Parallel reduction for the Dual Calculus</td>
<td>48</td>
</tr>
<tr>
<td>3.2</td>
<td>Duality for the arrow and quantifier types of DC</td>
<td>49</td>
</tr>
<tr>
<td>3.3</td>
<td>Syntactic actions for second-order DC</td>
<td>66</td>
</tr>
<tr>
<td>3.4</td>
<td>Formulas for calculation of image of reduction</td>
<td>67</td>
</tr>
<tr>
<td>3.5</td>
<td>Second-order type actions for Orthogonality Pairs</td>
<td>77</td>
</tr>
<tr>
<td>4.1</td>
<td>System F</td>
<td>99</td>
</tr>
<tr>
<td>4.2</td>
<td>Extension of System F with existential types</td>
<td>100</td>
</tr>
<tr>
<td>4.3</td>
<td>Extension of System F with Mendler induction</td>
<td>103</td>
</tr>
<tr>
<td>4.4</td>
<td>Two positive, one negative, and one mixed typeschemes (top to bottom)</td>
<td>105</td>
</tr>
<tr>
<td>4.5</td>
<td>Extension of System F with ordinary induction</td>
<td>107</td>
</tr>
<tr>
<td>4.6</td>
<td>The functorial actions for F(X) for terms f and i</td>
<td>108</td>
</tr>
</tbody>
</table>
LIST OF TABLES

4.7 The translation of the implicative fragment with ordinary induction into System F with Mendler induction .............................. 110
4.8 Translation of types in System F with Mendler induction to System F with ordinary induction .................................................. 114
4.9 Translation of terms in System F with Mendler induction to System F with ordinary induction .................................................. 115
5.1 The types and syntax of DC with Mendler induction ......................... 118
5.2 Translation of Intuitionistic Mendler induction into the Dual Calculus with Mendler induction .................................................. 123
5.3 Extension of the propositional DC with inductive types ..................... 127
5.4 Translation of Classical induction into Classical Mendler Induction ..... 130
5.5 The Mendler version of the mapping for continuations .................... 131
6.1 Parallel reduction for the Dual Calculus with Mendler induction ........ 138
6.2 Extension of duality to Mendler-inductive types ............................ 139
Chapter 1

Introduction

§1.1. Bridging Logic and Computer Science  One of the most fascinating aspects of Theoretical Computer Science is the elegant way in which it lays bare the relation between the acts of proving and programming. The Curry-Howard isomorphism, as the correspondence is known, was originally stated for combinatory logic and the simply typed lambda-calculus \((Curry \text{ and } Feys, 1958; \text{Howard}, 1980, \text{respectively})\) but the idea has been adapted to many other logics and their corresponding programming languages. The two domains linked, we can transpose results from one field to the other: e.g., a strongly-normalization result (every program stops) implies the consistency of the associated logic. Pragmatically and aesthetically, then, the Curry-Howard isomorphism matters—but how far can we take it?

1.1 The Curry-Howard Isomorphism

"The system was introduced in the context of proof theory (Girard, 1971), but it was independently discovered in computer science (Reynolds, 1974)."

Jean-Yves Girard et al. (1989)

§1.2. Formulas Are Types  It all began with a footnote—a side, seemingly unimportant, remark—in Curry’s introduction to combinatory logic: that the rules governing the combinators

\[
kxy = x \quad \text{and} \quad sxyz = (xz)(yz)
\]

were in some way related to Hilbert’s axioms in the sense that their types are

\[
k : A \rightarrow B \rightarrow A \quad \text{and} \quad s : (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C.
\]

\footnote{The book by Sørensen and Urzyczyn (2006) is a thorough introduction to the various aspects of the isomorphism.}
§1.3. Proofs Are Programs  This observation describes the first half of the isomorphism: formulas are types. Howard completed it by showing that, actually, in the simply typed lambda-calculus programs determine their typing derivations and these have exactly the same shape as some proof of the type read as a formula of propositional logic. For example:

\[
\frac{x : A, y : B}{x : A \vdash x : A} \quad \frac{A, B \vdash A}{A \vdash B \rightarrow A} \quad \frac{x : A \vdash \lambda y : B \cdot x : B \rightarrow A}{\vdash \lambda x : A. \lambda y : B \cdot x : A \rightarrow B \rightarrow A} \quad \frac{A \vdash B \rightarrow A}{\vdash A \rightarrow B \rightarrow A}
\]

The 1970s witnessed two seminal—much celebrated, oft confused—developments that impressed upon the power of the isomorphism: separated by an ocean, the American Computer Scientist John C. Reynolds (1974) created the polymorphic lambda-calculus, whilst the French logician Jean-Yves Girard (1971) created System F. The titles of their works, “Towards a theory of type structure” and “Functional application and cut-elimination in higher-order arithmetic”, leave no doubt to their differing aims; and, yet, their approaches are similar and the two systems synthesised therein are essentially the same. All we need is the correcting lenses of the Curry-Howard isomorphism to spot the connection between polymorphism and second-order universal quantification.

§1.4. Less Celebrated but Equally Useful Links  But the connection does not stop here. Some twenty-odd years after Girard and Reynolds, Mitchell and Plotkin (1988) completed the connection between second-order types and programming constructs: existential types correspond to abstract data-types. In an abstract data-type, the concrete implementation of a value is hidden (read, its type is hidden), and only certain operations apply. Furthermore, this structure is sufficiently expressive to model object-oriented programming—again, showing the power of Logical constructions in hitting at the right kind of interpretations for concrete, useful programming constructions.

There is a meta-strategy in the above argument. Logical connectives come in dual pairs: conjunction vs. disjunction, universal vs. existential quantification. Finding the corresponding programming construct to one connective immediately raises the question of which programming construct does its dual correspond to. Conspicuously gone from the list of connectives above is implication; if it corresponds to the type of functions, what does its dual, subtraction, correspond to? The answer, roughly, is to the concurrent counter-part of functions, co-routines (Crolard, 2004).

§1.5. The Elephant in the Curry-Howard Room  All the equivalences above were stated for Intuitionistic systems\(^2\). These systems had not even been defined until the 20th century. Classical Logic—that of Plato and Aristotle—remained, for a very long time, immune to all attempts to reframe it computationally. It turned out that, to do so, needed a complete change of perspective, one that went beyond the study of pure functions.

\(^2\)See, for example, the book by Troelstra and van Dalen (1988).
1.2 Classical Logic and Control

“The extraction of programs from classical proofs, which has long been considered as impossible, has raised a lot of interest since [...] the fact that control operators can be given types like \( \neg \neg F \rightarrow F \) was discovered by Griffin (1989).”

Jean-Louis Krivine (1994)

§1.6. Control In a landmark letter to the editor, Esdger Dijsktra made the case for avoiding direct use of the \texttt{go to} statement (Dijkstra, 1968). Upon finding this statement, the computer would not continue with the ensuing instruction, as normal, but instead reroute its execution context to wherever the statement told it to. Code with this operator made programmers less efficient and more prone to errors by obfuscating the static relations between the pieces of code and forcing them to rely more on its dynamic evolution—a task humans are ill-equipped for. The \texttt{go to} statement is an example of a control construct: it controls directly the execution path of the program. The letter was influential, and direct usage of control operators dwindled—and yet it is still extant. Why? Two reasons: power and efficiency. Direct access to control gives the program the ability to determine where exactly it needs to go next to better perform its task.

§1.7. Classical Programming Seen through the lenses of the Curry-Howard isomorphism, control operators are what we need to create logics that are Classical: they satisfy the law of the excluded middle, \( P \lor \neg P \), or (equivalently) satisfy the property of double-negation elimination, \( \neg \neg P \rightarrow P \), or (equivalently) Peirce’s law:

\[
(((P \rightarrow Q) \rightarrow P) \rightarrow P).
\]

This is the fundamental insight that we owe to Griffin (1989). He showed that Felleisen’s \( C \) operator’s type corresponded to the principle of double-negation elimination. Using that operator, he showed that (an idealized version of) the Scheme programming language with the call-with-current-continuation construct corresponded to a system of natural deduction with Peirce’s Law. And, he also completed the correspondence with the Classical principles enumerated above by showing how double-negation elimination and a disjunctive extension of the language with the law of the excluded middle could encode each other at the type and operational level. Independently, Filinski (1989) also noticed the connection between control (in its declarative continuation form) and Classical Logic.

§1.8. Further Developments These two results opened the proverbial floodgates and over the years many tweaks and extensions have been proposed; in common, they all share some relation to constructs wherein the flow of execution is interrupted. The \( \lambda \mu \) calculus of Parigot (1992b) is a Curry-Howard distillation of a sequent calculus like system, with

\footnote{The equivalences are not exact in that some systems may not have necessary structure (functional types, disjunctives types, etc.) to support all of the laws.}
modus ponens; de Groote (1994b) showed this system to be operationally equivalent to Griffin’s system. Further, de Groote (1995) also created a Classical calculus based around the notions of exceptions—another control construct.

§1.9. Duality  The next wave of disruption—as it were—came with the realization that, inherently, Classical Systems have built-in duality—a duality which is perfectly captured by Gentzen’s Sequent Calculus, LK (Gentzen, 1964). Firstly, Barbanera and Berardi (1996) introduced the symmetric lambda-calculus, a calculus based around duality of the connectives witnessed by the de Morgan laws. Curien and Herbelin (2000) introduced the $\lambda\mu\bar{\mu}$, almost the Curry-Howard equivalent of LK. This system was endowed with two abstractions and prioritizing the reduction of one over the other, yielded two different systems—the call-by-name and call-by-value variants—which were dual to one another. Wadler built up on this in his Dual Calculus (Wadler, 2003), providing more refined notions of call-by-name and call-by-value whence he extracted a reflection between his calculus and the associated continuation-passing-style transform. This calculus—in conversion rather than reduction guise—also had a very strong correspondence with the (conversion theory associated with the) $\lambda\mu$ calculus. Yet another LK-like calculus can be found in the work of Urban and Bierman (2001).

1.3 Mendler Induction

§1.10. Harnessing the Power of Types  Another area in which types show their importance is in preserving the safety invariants of a given language—strong normalization being a typical example. And one situation where strong normalization tends to go by the wayside is when one tries to consider inductive or recursive definitions—definitions that appeal to the very thing being defined. The most common way to guarantee strong normalization of a language with inductive definitions is by restricting these to special types formed from the closure of a positive typescheme. Positivity translates, at the level of models, to a monotone function. The intuition is that further applications of the typescheme (or its model interpretation) yield values imbued with more information. When performing induction on a value, these systems also take care of calling the induction on the sub-values of inductive type directly—thereby removing the responsibility and the possibility of error from the programmer. Mendler (1991) discovered that through an ingenious typing trick, involving nothing but universal second-order types, he could remove the possibility of circular inductive definitions whilst handing back to the programmer the power to chose where induction is called. Fundamental here was the ability of second-order types to enforce hiding invariants—as we already alluded to in the connection between these types and abstract data-types.

§1.11. Expressive Power  Mendler inductive-types can express all the functions that are catamorphisms, assuming the associated map function, responsible for the inductive

---

4 This in the sense of Meijer et al. (1991) although, since we are considering termination, we restrict the class of models under consideration to those where initial algebras and final co-algebras do not, in general, coincide.
invocation on the smaller inductive values, is expressible in the host language. This be-
cause with the map, the inductive invocation, and the intended inductive step, Mendler
induction provides to its inductive step all the elements involved in the reduction of cata-
morphisms.

§1.12. Negative Typeschemes and Higher-order Types Another aspect where Mendler
induction affords greater generality is on the type-schemes that it inducts on. The posi-
tivity requirement is necessary for otherwise it is impossible to define the associated map.
Mendler induction is not dependent on any outside structure; so it can induct on arbi-
trary typeschemes—a fact due, independently, to Uustalu and to Matthes (Uustalu and
Vene, 1999; Matthes, 1999a). Model-theoretically, one is forced to consider monotone
extensions of the underlying interpretation of a typescheme in order to achieve the nec-
essary monotonicity. This incredible generality made Mendler induction important in
the study of recursion at higher-order types (Uustalu and Vene, 2005; Ahn and Sheard,
2011). A final example of the applicability of Mendler induction is in the derivation
of proof tactics for statements defined (co-) inductively where the underlying statement
makes contra-variant use of the induction hypothesis (Hur et al., 2013).

1.4 Thesis

§1.13. Statement This thesis should be viewed on two levels: on the level of Princi-
ples and on the level of Pragmatics. As a principle, we contend that the Curry-Howard
isomorphism should not be reserved for post hoc analysis of programming languages
but should itself be seen as a tool for synthesizing powerful programming languages and
interesting constructs from Logical systems. Pragmatically, we assert that Mendler in-
duction is a general inductive principle that is compatible with control (Devesas Campos
and Fiore, 2016). How?

§1.14. From LK to DC We start (chapter 2) by reviewing Gentzen’s Sequent Calculus,
LK; as a Logical system, it derives its deductive power directly from its structure. It also
exhibits an interesting form of duality that is embodied by—as we will see—two of Com-
puter Science’s main notions: computation and continuation. Then, we present the Dual
Calculus in detail together with second-order types, and we extend it with subtractive
types—completing, further, the duality noted by Gentzen.

Chapter 3 shows that the system thus created has many of the properties we expect
from a well-behaved programming language: it is subject preserving; it satisfies the substi-
tutivity properties; and—via a long realizability argument, that goes deep into the details
of the language—we show that the associated reduction rules are strongly normalizing.

§1.15. Stopping at Mendler Our goal of adding Mendler induction to the Dual Calculus
begins with a derivation of Mendler induction from operational considerations (chap-
ter 4) and showing how it can be put to practical use. We then prove formally that
Mendler induction can simulate induction; but also that the ordinary notion of induction
can—with some extra complexity—be used to simulate Mendler induction.
§1.16. **Arriving at Control and Induction**  
Our destination, and the original contribution of this thesis, is then within reach: the definition of an extension to the Dual Calculus with Mendler induction (chapter 5). We will show that the system thus defined subsumes a host of other programming languages: the lambda-calculus, System F with Mendler Induction, plus another extension of the Dual Calculus with induction for only positive typeschemes. And, all the while (cf. chapter 6), the system is still well-behaved: mirroring what we did for the regular Dual Calculus, we show that the system is subject preserving, respects substitutivity, and we show how to expand our model of strongly normalizing phrases to accommodate Mendler induction.
Chapter 2

The Dual Calculus

“If ⊃-IS and [...] ⊃-IA are excluded, the calculus \( LK \) is dual in the following sense: If we reverse all sequents of an \( LK \)-derivation [etc.], then another \( LK \)-derivation results.”

Gerhard Gentzen, *Investigations into Logical Deduction* (in Szabo, 1969, §2.4)

2.1 Gentzen’s \( LK \)

§2.1. Gentzen’s Systems  To call Gerhard Gentzen’s “Investigations into Logical Deduction” (Gentzen, 1964) simply an important piece of work would be a untenable understatement. Though published in 1935, this paper has set the standard for all syntactical developments in Logic ever since: it moved the world away from Hilbert’s deduction systems, giving us a new way to present logical arguments more intuitive, more beautiful, but in no way less precise; it presented a new (structural) way of looking at Classical proofs that did not rely directly on any postulates, such as the Law of the Excluded Middle; and it presaged the importance of strong normalization results and normal forms in analysing logical systems, culminating in a ground-breaking consistency result. Case in point for the staying power of this work: some eighty years since its first publication, here is another thesis based on one of the *Investigations*’ systems.

In fact, Gentzen described six different systems in that one paper—though we nowadays see them as three systems, each in Intuitionistic and Classical variants: Natural Deduction, Sequent Calculus, and a formalization of Hilbert’s deduction system—the latter being only used in relating the logical properties of the first two with the then more accepted method of doing deduction. Natural Deduction was the system that Gentzen was interested in the most—in that it better reflected the actual practice of mathematical proof. However, the sequent calculus—being better-behaved syntactically—is more amenable to formal analysis. Gentzen’s famed *Hauptsatz* (main theorem) of the Sequent

\[ 1 \text{ This because each sequent contains within itself all the information necessary to prove its validity: all the assumptions and all possible consequences are right there, in the sequent. The same is not true of Natural Deduction where the dischargeable assumptions hover around, as it were, and are very much dependent on context.} \]
Calculus has as corollaries the consistency of the calculus and (conveniently) the establishment of the sub-formula property—that every provable formula can be proven using only assumptions that are sub-formulas of the target sequent. The same properties can be proven for Natural Deduction by translating it into the Sequent Calculus.

If we want to study and prove things about a system that is related (via the Curry-Howard isomorphism) to Classical Logic—such as the Dual Calculus (Wadler, 2003)—it stands to reason that the Sequent Calculus in its Classical flavour, LK, is a good place to start.

§2.2. Sequents  The central concept in the Sequent Calculus is that of a sequent. Sequents relate needed assumptions and their possible consequences. They are represented by Gentzen as

\[ \Gamma \rightarrow \Delta \]

where \( \Gamma \) and \( \Delta \) are finite sequences of formulas. For any such sequent the slogan is: whenever all the formulas (assumptions) in \( \Gamma \) are true, one of the formulas (consequences) in \( \Delta \) shall be true.

In Classical Logic, it is true of every formula \( A \) that either \( A \) itself or its negation, \( \neg A \), must be true. Readily, then, we have that every sequent of the form \( \rightarrow A, \neg A \) must be valid. (Note the empty sequence to the left of the \( \rightarrow \) sign.) Conversely, the sequent \( A, \neg A \rightarrow \Delta \) also conforms to the reading above (vacuously, by not fulfilling the preconditions) and is also valid.

The rest of LK is the collection of deduction rules from which we determine valid sequents—and which we shall now turn our attention to.

§2.3. Structural Rules  The interpretation above alone informs the definition of a series of rules relating solely to the structure of sequents. They are therefore termed structural and table 2.1 summarises them. We represent lists of formulas by capital Greek letters and individual formulas by capital Roman letters.

The first rule, the one from which every derivation in the sequent calculus commences, is the Identity rule; it says—rather fundamentally—that if we have assumed that a formula holds then we can conclude that that formula holds. The Thinning rules say that we can always expand the list of assumptions, or the list of possible consequences: if everything in \( D; \Gamma \) holds, then everything in \( \Gamma \) holds; whence, if we already knew that this set of assumptions caused something in \( \Theta \) to hold, then it must be the case that assuming that \( D; \Gamma \) holds still forces something in \( \Theta \) to hold. Dually, if the whole of \( \Gamma \) holding makes something in \( \Theta \) hold, then it must also make something in \( \Theta, D \) hold. Contraction states that knowing several times that one assumption holds is the same as just knowing that it holds once—and likewise, dually, for consequences. Interchange says that the order of our assumptions/consequences is irrelevant. Together, Thinning, Contraction, and Interchange allow us to express the sequences of formulas in the definition of a sequent as simply finite sets.

§2.4. The Cut Rule  The final structural rule relates to how sequents can be, in some sense, composed. Assume we have a sequent \( \Gamma \rightarrow \Theta, D \) and a sequent \( \Delta, D \rightarrow \Lambda \). If everything in \( \Gamma \) holds, either something in \( \Theta \) is true or \( D \) is true. In the latter case, if we
GENTZEN’S LK

Identity:

\[ D \rightarrow D \]

Thinning:

\[
\begin{align*}
\Gamma \rightarrow \Theta & \quad \Gamma \rightarrow \Theta \\
D, \Gamma & \rightarrow \Theta & \Gamma \rightarrow \Theta, D
\end{align*}
\]

Contraction:

\[
\begin{align*}
D, D, \Gamma & \rightarrow \Theta & \Gamma \rightarrow \Theta, D, D \\
D, \Gamma & \rightarrow \Theta & \Gamma \rightarrow \Theta, D
\end{align*}
\]

Interchange:

\[
\begin{align*}
\Delta, D, E, \Gamma & \rightarrow \Theta & \Gamma \rightarrow \Theta, E, D, \Lambda \\
\Delta, E, D, \Gamma & \rightarrow \Theta & \Gamma \rightarrow \Theta, D, E, \Lambda
\end{align*}
\]

Cut:

\[
\begin{align*}
\Gamma \rightarrow \Theta, D & \quad D, \Delta \rightarrow \Lambda \\
\Gamma, \Delta & \rightarrow \Theta, \Lambda
\end{align*}
\]

Table 2.1: The identity and structural rules of Gentzen’s LK (Gentzen, 1964)

also know that everything in \( \Delta \) holds, we can conclude something in \( \Lambda \) holds. Joining the assumptions, we get that something in the list of possible consequences \( \Theta, \Lambda \) holds.

The Cut rule is by far and away the most striking of the rules of the Sequent Calculus—it is certainly plausible but by no means intuitive. From the point of view of provability in Propositional Logic this rule is, in fact, redundant—the Hauptsatz being the formal statement of this. However, from a programming point of view, where the main theme is reduction, it is essential. It is the Cut rule that codifies the way different proofs (read, programs) can be plugged together to generate more complicated ones.

§2.5. Structural Duality The essence of duality in the Sequent Calculus (and proper derivatives) materializes in the structural rules. The interpretation that guided our choice of the rules above is defined using two dualities: antecedent–consequent, and all–any. Had we chosen to interpret a list of formulas on the right of a sequent as all of them holding, the thinning rule on the right would no longer be valid. As it stands, we have that in the presence of an appropriate duality of formulas, any calculus based on the structural fragment of the Sequent Calculus is itself dual.

**Proposition 1.** Let \( \sim \) be an involution of formulas —i.e. \( A \sim = A \)—lifted to sequences of antecedents and of consequents in a point-wise manner. If a sequent \( \Gamma \rightarrow \Delta \) is derivable in the structural fragment of the Sequent Calculus, so is its dual \( \Delta \sim \rightarrow \Gamma \sim \).

**Proof.** By induction on the structural rules:

The identity rule asserts \( D \rightarrow D \)—and, also, \( D \sim \rightarrow D \sim \).

If a sequent \( \Gamma, D \rightarrow \Theta \) is proven using left thinning then we have that \( \Gamma \rightarrow \Theta \) is provable, and, by induction hypothesis, so is \( \Theta \sim \rightarrow \Gamma \sim \); from, here we get via right thinning that \( \Theta \sim \rightarrow \Gamma, D \sim \). Right thinning follows dually.
THE DUAL CALCULUS

If, instead, a sequent \( \Gamma \rightarrow \Theta, D, E, \Lambda \) is derived from an application of right exchange (and dually for left exchange), it must be because \( \Gamma \rightarrow \Theta, E, D, \Lambda \) has been previously derived, and so can \( \Theta^o, E^o, D^o, \Lambda^o \rightarrow \Gamma^o \), by the induction hypothesis. Finally, using the left exchange rule, we can derive \( \Theta^o, D^o, E^o, \Lambda^o \rightarrow \Gamma^o \).

The cut rule presents only slightly more difficulties simply because the application of the induction hypothesis does not leave the formulas in the right order to apply the cut rule. This can be corrected using left- and right-interchanges. Diagrammatically we have:

\[
\begin{align*}
D, \Delta & \rightarrow \Lambda & \quad & \begin{array}{ll}
\text{IH} & \Gamma \rightarrow \Theta, D \\
\text{IH} & \Theta^o, D^o \rightarrow \Gamma^o
\end{array} \\
\Lambda^o & \rightarrow D^o, \Delta^o & \begin{array}{ll}
R\text{-interchanges} & \text{L-interchanges}
\end{array} \\
\Lambda^o \rightarrow \Delta^o, D^o & \\
\Lambda^o, \Theta^o \rightarrow \Delta^o, \Gamma^o & \text{Cut}
\end{align*}
\]

\[
\begin{align*}
\text{L\&R-interchanges} \\
\Theta^o, \Lambda^o \rightarrow \Gamma^o, \Delta^o
\end{align*}
\]

§2.6. Logical Rules  

We now move on to the second-order propositional rules laid out by Gentzen (he called them operational). These permit making deductions using conjunctions, disjunctions, negations, implications, and universal and existential quantifications. The rules are put forth in table 2.2 and are again separated in their left- and right-actions on formulas and their principal connectives. We have retained the nomenclature presented by Gentzen (1964), prefixing left rules with ‘A’ (for ‘antecedent’) and ‘S’ (for ‘succedent’). Lower-case letters stand for type meta-variables, \( F \) for some formula-scheme: a formula in which certain positions can instantiated (via substitution) with arbitrary formulas. We denote the instantiation of a formula-scheme \( F \) with the type represented by a meta-variable \( a \) by \( Fa \).

§2.7. Conjunction  

Beginning with conjunctions, \( A \& B \), if under a set of hypothesis two propositions become possibly true, then so does their conjunction—this is how one introduces conjunctions in consequences. Conversely, if from a proposition \( A \) we derive that some consequences may hold, then from the conjunction \( A \& B \) of \( A \) with some \( B \) we can deduce that those same consequences may hold—as the conjunction should imply its constituent formulas. (Note the non-standard notation \& which we will keep here for historical accuracy but which we shall not follow later on.)

§2.8. Disjunction  

The behaviour of disjunction is dual to that of conjunction: a disjunction \( A \vee B \) holds whenever we have that \( A \) or \( B \) individually hold. Conversely, if from both \( A \) and \( B \) we derive the same set of consequences we can derive that same set via the disjunction.

§2.9. Negation  

The slogan for sequents \( \Gamma \rightarrow \Delta \) is that whenever all formulas in \( \Gamma \) hold, then so does one of those in \( \Delta \). So, if additionally we know that some \( A \) in \( \Delta \) does not
GENTZEN’S LK

&-IS:

\[
\Gamma \rightarrow \Theta, A \quad \Gamma \rightarrow \Theta, B
\]
\[
\Gamma \rightarrow \Theta, A \land B
\]

&-IA:

\[
A, \Gamma \rightarrow \Theta \quad B, \Gamma \rightarrow \Theta
\]
\[
A \land B, \Gamma \rightarrow \Theta
\]

\lor-IA:

\[
A, \Gamma \rightarrow \Theta \quad B, \Gamma \rightarrow \Theta
\]
\[
A \lor B, \Gamma \rightarrow \Theta
\]

\lor-IS:

\[
\Gamma \rightarrow \Theta, A \quad \Gamma \rightarrow \Theta, B
\]
\[
\Gamma \rightarrow \Theta, A \lor B
\]

\forall-IS:

\[
\Gamma \rightarrow \Theta, Fa
\]
\[
\Gamma \rightarrow \Theta, \forall r. Fr
\]

\exists-IA:

\[
Fa, \Gamma \rightarrow \Theta
\]
\[
\exists r. Fr, \Gamma \rightarrow \Theta
\]

\neg-IS:

\[
A, \Gamma \rightarrow \Theta
\]
\[
\Gamma \rightarrow \Theta, \neg A
\]

\neg-IA:

\[
\Gamma \rightarrow \Theta, A
\]
\[
\neg A, \Gamma \rightarrow \Theta
\]

\supset-IS:

\[
A, \Gamma \rightarrow \Theta, B
\]
\[
\Gamma \rightarrow \Theta, A \supset B
\]

\supset-IA:

\[
\Gamma \rightarrow \Theta, A \quad B, \Delta \rightarrow \Lambda
\]
\[
A \supset B, \Gamma, \Delta \rightarrow \Theta, \Lambda
\]

In the two aforementioned rules the meta-variable \(a\) must not appear free in \(Fr\), nor in \(\Gamma\), nor in \(\Delta\).

\forall-IA:

\[
Fa, \Gamma \rightarrow \Theta
\]
\[
\forall r. Fr, \Gamma \rightarrow \Theta
\]

\exists-IS:

\[
\Gamma \rightarrow \Theta, Fa
\]
\[
\Gamma \rightarrow \Theta, \exists r. Fr
\]

\neg-IA:

\[
\Gamma \rightarrow \Theta, A
\]
\[
\neg A, \Gamma \rightarrow \Theta
\]

\supset-IS:

\[
A, \Gamma \rightarrow \Theta, B
\]
\[
\Gamma \rightarrow \Theta, A \supset B
\]

\supset-IA:

\[
\Gamma \rightarrow \Theta, A \quad B, \Delta \rightarrow \Lambda
\]
\[
A \supset B, \Gamma, \Delta \rightarrow \Theta, \Lambda
\]

Table 2.2: The operational rules of Gentzen’s LK (Gentzen, 1964)
THE DUAL CALCULUS

hold (i.e., \( \neg A \) holds) then it must be some of the other formulas in \( \Delta \) that holds. This is exactly what the introduction rule of negation on the left does. Conversely, if a set of assumptions \( \Gamma \) requires \( A \) to hold in order to make something in \( \Delta \) hold, then either \( A \) holds and something in \( \Delta \) holds, or it is \( \neg A \) that holds.

§2.10. Implication  In the ancient tradition, Gentzen represents implication by \( A \supset B \). If \( B \) holds under an assumption \( A \) then it must be that the implication \( A \supset B \) holds. Conversely, if some \( A \) holds and an implication \( A \supset B \) also holds then every possible consequence from \( B \) holds. The left implication rule states this but in the more general setting where sequents have additional antecedents and consequences. We can understand this rule, also, as a reworking of the cut rule—we don’t require the cut formulas to be the same, they can be any \( A \) and \( B \) (resp.) but \( A \) must imply \( B \).

§2.11. Universal Quantification  A universal statement manifested by a formula \( \forall r.Fr \) is derivable as a consequence exactly when all its instances \( Fa \) are derivable. In other words, the knowledge of the exact instance must be irrelevant to the truth of the statement. Testing this instance-by-instance would be impossible; instead, we require that the formula scheme\(^2\) be provable for some arbitrary variable \( r \); this variable must not appear free (that is outside the scope of another quantifier) in the remaining formulas of the sequent.\(^3\)

Introduction on the left is much simpler: if a sequent is valid when a particular instance of \( F \) is used as an assumption then all conclusions drawn from it can also be drawn from the universal quantification of \( F \)—as this, in turn, implies the instance.

To see the importance of the restriction for the right rule, consider the following invalid derivation of the statement that any abstract instance \( r \) implies its own negation:

\[
\begin{align*}
\frac{r \vdash r \quad \neg r \vdash \neg r}{r \vdash \forall s.s} & \\
\frac{r \vdash \forall s.s \quad \forall s.s \vdash \neg r}{r \vdash \neg r}
\end{align*}
\]

§2.12. Existential Quantification  Dually, we can easily derive that an existential quantification \( \exists r.Fr \) is a consequence in some sequent if we can prove that in the same sequent, one of its instances \( Fa \) is a consequence.

However, to prove that a sequent follows from some existentially quantified hypothesis, we need to assert that knowledge of the particular instance that makes the sequent valid is used nowhere in the derivation—only then is the instance truly abstract. Like for the universal case, this is done by instantiating \( F \) with an arbitrary meta-variable that does not appear (free) anywhere else in the sequent.

---

\(^2\)Following the terminology of \text{Gentzen} (1964).

\(^3\)This idea is crucial to the definition of Mendler Induction in a Classical setting that we offer later on (chapter \text{5}).
§2.13. Classical Axioms in LK  As a way of better understanding Sequent Calculus we shall now see how several Classical Logic entailments are derivable in it. Understanding these will pay off later when we turn to how these Classical features translate into the programming world. We begin with the law of the excluded middle. We have argued before that it should be derivable in LK for any proposition \( A \); we can do this thus:

\[
\begin{align*}
A & \rightarrow A \\
\rightarrow A, \neg A & \\
\rightarrow A \lor \neg A, \neg A & \\
\rightarrow A \lor \neg A & \\
\end{align*}
\]

Double-negation elimination is deceptively simply:

\[
\begin{align*}
A & \rightarrow A \\
\rightarrow A, \neg A & \\
\neg\neg A & \rightarrow A \\
\end{align*}
\]

Finally, we have the de Morgan laws:

\[
\begin{align*}
\neg A & \rightarrow \neg (A \lor B) \\
\neg A & \rightarrow \neg (A \rightarrow B) \\
\neg (A \lor B) & \rightarrow \neg A \\
\neg (A \land B) & \rightarrow \neg A \\
\end{align*}
\]

§2.14. True and False  The language of LK makes no mention of truth or falsity. This is because these concepts can be simulated by the other constructs of the language. The key insight is to look at falsity as something from which all other propositions follow, and—dually—to truth as something that follows from everything.

**Proposition 2.** Take an arbitrary proposition \( A \). For any proposition \( D \) we can derive

\[
A \land \neg A \rightarrow D \quad \text{and} \quad D \rightarrow A \lor \neg A.
\]
THE DUAL CALCULUS

\[
\begin{align*}
X^\circ &= X \\
(A \land B)^\circ &= A^\circ \lor B^\circ \\
(\forall r. Fr)^\circ &= \exists r. (Fr)^\circ \\
(A \lor B)^\circ &= A^\circ \land B^\circ \\
(\exists r. Fr)^\circ &= \forall r. (Fr)^\circ
\end{align*}
\]

Table 2.3: The duality of formulas in LK.

Proof. The two derivations are symmetric from each other; for truth we reuse the derivation of the Law of the Excluded Middle:

\[
\begin{align*}
\Gamma \to A \\
\hline
\Gamma, \neg A \\
\hline
\Gamma, \neg A, \neg A \\
\hline
\Gamma, \neg A, \neg A, A \lor \neg A \\
\hline
\Gamma, A \lor \neg A, \neg A \\
\hline
\Gamma, A \lor \neg A, A \lor \neg A \\
\hline
\Gamma, A \lor \neg A \lor A \\
\hline
\Gamma, D
\end{align*}
\]

\[
\begin{align*}
\Gamma \to A \\
\hline
\Gamma \to A, \neg A \\
\hline
\Gamma \to A \lor \neg A, \neg A \\
\hline
\Gamma \to A \lor \neg A, A \lor \neg A \\
\hline
\Gamma \to A \lor \neg A
\end{align*}
\]

\[\square\]

§2.15. Duality We can make the duality in proposition \[\square\] stronger—and give body to Gentzen’s remark at the start of the chapter—by considering the duality of formulas \((-)^\circ\) in table 2.3. (Implication is—as per the quote at the start of the chapter—omitted from this particular discussion.)

Proposition 3 (Duality of LK). If a sequent \(\Gamma \to \Theta\) is derivable in LK then so is its dual \(\Theta^\circ \to \Gamma^\circ\).

Proof. The proof is by rule induction. The duality operator \((-)^\circ\) falls within the conditions of proposition \[\square\] and, therefore, for the identity and structural rules we may reuse the proof of that theorem.

Let’s start with conjunctions \(A \land B\). The dual of the right conjunction rule is the left disjunction rule, as per \[\square\].

\[
\frac{(\Gamma \to \Theta, B)}{B^\circ, \Theta^\circ \to \Gamma^\circ} \quad \frac{(\Gamma \to \Theta, A)}{A^\circ, \Theta^\circ \to \Gamma^\circ}
\]

\[
\frac{B^\circ \lor A^\circ, \Theta^\circ \to \Gamma^\circ}{\Theta^\circ \to \Gamma^\circ}
\]

For the left and right rules and we have, respectively:

\[
\frac{(A, \Gamma \to \Theta)}{\Theta^\circ \to \Gamma^\circ, A^\circ} \quad \frac{(B, \Gamma \to \Theta)}{\Theta^\circ \to \Gamma^\circ, B^\circ}
\]

\[
\frac{\Theta^\circ \to \Gamma^\circ, B^\circ \lor A^\circ}{\Theta^\circ \to \Gamma^\circ}
\]

\[\square\]

\[\square\]

\[\square\]

\[\square\]

\[\square\]Applications of the induction hypothesis are not, strictly speaking, derivations from the rules of LK. To make this distinction clear, we surround the assumptions to which we apply the induction hypothesis by parenthesis.
Duals for disjunctions are proven dually:

\[
\begin{align*}
\frac{(B, \Gamma \rightarrow \Theta)}{\Theta^o \rightarrow \Gamma^o, B^o} & \quad \frac{(A, \Gamma \rightarrow \Theta)}{\Theta^o \rightarrow \Gamma^o, A^o} \\
\frac{\Theta^o \rightarrow \Gamma^o, B^o \& A^o}{(\Gamma \rightarrow \Theta, A)} & \quad \frac{(\Gamma \rightarrow \Theta, B)}{B^o, \Theta^o \rightarrow \Gamma^o}
\end{align*}
\]

Negations are their own duals:

\[
\begin{align*}
\frac{(A, \Gamma \rightarrow \Theta)}{\Theta^o \rightarrow \Gamma^o, A^o} & \quad \frac{(\Gamma \rightarrow \Theta, A)}{A^o, \Theta^o \rightarrow \Gamma^o} \\
\frac{\neg A^o, \Theta^o \rightarrow \Gamma^o}{\neg A^o, \Theta^o \rightarrow \Gamma^o} & \quad \frac{\Theta^o \rightarrow \Gamma^o, \neg A^o}{\Theta^o \rightarrow \Gamma^o}
\end{align*}
\]

The dual of the right rule of universal quantification is the left existential quantification rule. To use the latter we need to respect the restriction on the meta-variable. However, this is the same restriction respected by the right universal quantification rule and, luckily, the dualization operation respects the restriction as it does not introduce any new (meta-)variables into the mix. The proof for the left rule is even simpler as it does not need to account for this restriction.

\[
\begin{align*}
\frac{(\Gamma \rightarrow \Theta, Fa)}{(Fa)^o, \Theta^o \rightarrow \Gamma^o} & \quad \frac{(Fa, \Gamma \rightarrow \Theta)}{\Theta^o \rightarrow \Gamma^o, (Fa)^o} \\
\frac{\exists r.(Fr)^o, \Theta^o \rightarrow \Gamma^o}{\Theta^o \rightarrow \Gamma^o, \exists r.(Fr)^o}
\end{align*}
\]

Proofs for existential quantification are the exact duals. □

§2.16. Cut-elimination Gentzen introduced \( LK \) as an intermediate step to prove that his Natural Deduction system was consistent. He did this by embedding the latter in the former, subsuming derivation in the former by derivation in the latter, and proving that \( LK \) itself was consistent. By consistency of \( LK \) it is meant that the empty sequent \( \rightarrow \) is not derivable—for if that were the case, by thinning, we could obtain a proof \( \rightarrow D \) of every proposition; and, conversely, should every proposition be derivable, we could reach the empty sequent thusly:

\[
\begin{align*}
\rightarrow D & \quad \rightarrow \neg D \quad \neg D \rightarrow \\
\end{align*}
\]

Central here is the role of the cut-rule: it is the only rule that can have the empty sequent as a consequence. Gentzen’s main theorem—the \emph{Hauptsatz}, or the cut-elimination theorem—is exactly this: that every derivable formula in \( LK \) is derivable without cuts—from which consistency follows as a corollary.

\textbf{Theorem 1} (Hauptsatz, \cite{Gentzen1964}, §2.5). Every [...] \( LK \)-derivation can be transformed into an [...] \( LK \)-derivation that has the same endsequent and in which a “cut” does not occur.
2.2 Wadler’s Dual Calculus

§2.17. From Logic to Programming. It is high time to look at $LK$ from the point of view of Computer Science. One system with that exact goal is the Dual Calculus (DC) of Wadler (2003). This language will be the basis for the all the work presented henceforth. In this section we will introduce the dual fragment of Wadler’s original calculus. Wadler’s full system also included—much like Gentzen’s presentation of $LK$—implications but not their dual, subtractions.

2.2.1 Syntax

§2.18. Types. Starting with types, we essentially have the propositional fragment of $LK$ (see table 2.4). Type variables are denoted by capital Roman letters ($X, Y, ...$). We identify conjunction with products—having pairings and projections—and disjunction with co-products. We recover the usual notation of $\land$ for conjunction. Intuitively, negation should be seen as the type of continuations reified as values—this shall be made clearer when we discuss the typing rules for this calculus.

§2.19. The Left/Right Divide. Proceeding down to the term level, operational right rules—to use the terminology of Gentzen—are very easily seen to correspond to the introduction rules in Natural Deduction. Intuitively, they tell us how to compute complex proofs from simpler ones: e.g., a conjunction of two proofs is just the pairing of the proof of each component. Left (operational) rules, accordingly, stand for elimination rules, and their computational meaning is more easily seen for disjunction: to eliminate a disjunction $A \lor B$ we must say what shall be done if the value comes from the first disjunct, and what shall be done if it comes from the second disjunct—i.e., we must provide a continuation for each possible value. The Dual Calculus comprises, then, not one but two (mutually dual) calculi—one of computations and one of continuations. Henceforth, we shall reserve the term ‘term’ to computation terms, and ‘co-terms’ for continuation terms. Whenever we wish to refer to an arbitrary term of the language we will use the term ‘phrase’. The abstract syntax of DC is represented in table 2.5.

§2.20. Cuts—The Bridge Between. With computations and continuations in hand, we need a way to join the two together. Given a computation $t$ and a continuation $k$, the execution of the continuation on the former is called a statement, or a cut, and is represented by

$$t \cdot k.$$
The name ‘cut’ hints at the connection it has with the cut rule of $LK$. In DC, cuts are the only entities that have any sort of executional semantics. Computations and continuations do not indicate, per se, any sort of execution until they are linked via a cut. The relationship may be summarised as

$$\text{Terms} + \text{Co-terms} = \text{Cuts}$$

much as the celebrated computer science “equation”

$$\text{Data-structures} + \text{Algorithms} = \text{Programs}.$$ 

§2.21. Variables  The most basic syntactic entity are variables. They are related to the identity rules in $LK$ and, similar to them, they are foundational in the Dual Calculus. That being said, due to the more strict left–right division in DC, we have two types of variables: term variables, or simply variables, which we shall represent with lower-case Roman letters ($x$, $y$, etc.); and, co-term variables, or more commonly co-variables, which we will represent by Greek lower-case letters ($\alpha$, $\beta$, etc.).

§2.22. Abstractions  Given that we have variables in DC, it is natural to ask for a way to abstract them out—i.e. provide some mechanism to say how they can be replaced by
THE DUAL CALCULUS

some term in a cut. The Dual Calculus has an abstraction operator denoted by $\overline{x} \cdot (c)$.

The variable on the left is the variable to be abstracted. After receiving an argument $t$ via a cut, it continues by replacing the variable by $t$ in $c$—yielding $c\frac{t}{x}$—and running the resulting statement. This construct, by the aforementioned intuitive semantics, expects—or receives—computations and is therefore a continuation. Dually, there is a co-abstraction construct,

$\overline{\alpha} \cdot (c)$

that indicates a computation where co-variable $\alpha$ can be replaced by some continuation. It is a term for it must be paired with a co-term for execution to continue.

§2.23. Conjunction Operators Conjunctions are naturally related to products (though not necessarily in any categorical sense). Given two computations, $t$ and $t'$, they can be paired to provide a computation $\langle t, t' \rangle$ encapsulating the values of both—much like a proof of a conjunction can be had by pairing proofs for each conjunct. Given a pair of terms, if we have a continuation $k$ compatible with the first, we can naturally define a continuation $\text{fst}[k]$ on pairs which projects the first element out of the pair and continues by passing it on to $k$—and likewise for the second projection $\text{snd}[k]$.

§2.24. Disjunction Operators Dually, disjunction can be seen as a sort of co-product. Computations of this kind are generated by taking computations $t$ and then injecting them—either via $\mu_1(t)$ or $\mu_2(t)$—as needed on either side of the disjunction. To continue from such values, we need to know how to continue from each possibility generated from the injections—so, given two continuations $k$ and $k'$, we form a new continuation $[k, k']$ which inspects whichever argument was passed to it to see if it was created using the first or the second injection, strips the underlying value, and passes it on to the corresponding continuation.

§2.25. Negation Operators Negation is not as intuitive as a programming language type as the previous constructs. We can understand a computation $\text{not}(k)$ of negation type as encapsulating a suspended continuation $k$ so that it can be passed around as a computation. To be able to use these, there is a (dual) operator $\text{not}[t]$ which encapsulates computations $t$ as continuations. Executing the cut of an encapsulated continuation with an encapsulated computation is a matter of peeling away the encapsulation and proceeding with the execution in the natural order.

2.2.2 Typing

§2.26. Judgements We have three types of phrases in the Dual Calculus and, accordingly, three types of judgements:

$\Gamma \vdash t : A | \Delta$

\footnote{For readability we write the name of all syntactic operators and abstractions on the left. In Wadler's presentation, co-term operations appear on the right of their arguments. To the writer, the loss of lexical (though not syntactical) duality seems like an acceptable casualty in exchange for the ability to read programs from left to right.}
for terms;
\[ \Gamma \vdash k : A \rightarrow \Delta \]
for co-terms; and,
\[ \Gamma \vdash c \rightarrow \Delta \]
for cuts.

Note the symmetry in the syntax of judgements for terms and judgements for co-terms. Each phrase of the language is surrounded by a typing context of variables \( \Gamma \) on the left, and a typing co-context \( \Delta \) on the right. They are finite assignments of types to (co-) variables:

\[ x_1 : A_1, \ldots, x_n : A_n \quad \text{and} \quad \alpha_1 : A_1, \ldots, \alpha_m : A_m, \]

respectively. We place them on the left and on the right to emphasize the view that variables (read, assumptions) are inputs; and co-variables (read, conclusions) are where we output. This sense is related to the continuation-passing style (CPS) of programming wherein returning a value is construed with the calling upon of the continuation.

§2.27. Structural Rules  We start with the structural typing rules (table 2.6). The most basic rules are the identity rules of which we have two, one for each type of variable. Phrases and judgements are interpreted up-to \( \alpha \)-equivalence. The Thinning, Interchange, and Contraction rules express the set-like nature of contexts. In particular, it is redundant to have multiple assumptions, say \( x \) and \( y \), of the same formula \( A \); so we can just substitute, \( p[x/y] \), any instance of \( y \) by \( x \) in any phrase \( p \). Finally, the Cut rule types cut phrases. Cuts themselves have no type but are only well-formed whenever they connect terms and co-terms of equal types—under the proviso that their associated typing (co-) contexts are the disjoint.

§2.28. Operational Typing Rules  The operational typing rules (table 2.7) should be fairly obvious under the intuitive interpretation we have given to the different phrases of DC. Pairing of terms gives a term whose type is the conjunction of the types of the terms. Given a continuation \( k \) of type \( A \), the continuation \( \text{fst}[k] \) is a continuation of type \( A \land B \) (for some \( B \))—and, conversely, for \( \text{snd}[k] \). Injections have disjunction types for terms; the operator \( [k, k'] \) continues from a disjunction whenever \( k \) and \( k' \) continue from the respective disjuncts. Finally, the negation operators take terms to co-terms of negation type, and co-terms to terms of negation type.

§2.29. Abstraction Typing Rules  The tricky bit of DC and its typing rules lies in the abstractions (cf. table 2.8). It can be said that if negation turns terms into co-terms, and vice-versa, then abstractions (resp. co-abstractions) turn cuts into co-terms (resp. terms). If we have a cut that depends on a co-variable \( \alpha \) then the process of abstracting \( \alpha \) means that \( \alpha \) is something that can be replaced by other co-terms of the same type—i.e., we can pass to it (via a cut) any co-term of the same type. Dually, abstracting on a variable yields a co-term of that type—for it can be coupled, via a cut, with other terms of the same type.

\[ \text{Cf. §2.6} \]
THE DUAL CALCULUS

Identity:

\[
\begin{align*}
x : A & \vdash x : A \\
\end{align*}
\]

\[
\begin{align*}
\alpha : A & \vdash \alpha : A
\end{align*}
\]

Thinning:

\[
\begin{align*}
\Gamma & \vdash t : A | \Delta \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, x : A, \Gamma' & \vdash t : A | \Delta \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, k : A & \vdash \Delta \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, x : A, \Gamma' & \vdash k : A | \Delta \\
\Gamma & \vdash c : \Delta \\
\Gamma, x : A, \Gamma' & \vdash c : \Delta \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, x : A, \Gamma' & \vdash c : \Delta \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, x : A, \Gamma' & \vdash c : \Delta \\
\end{align*}
\]

\[
\begin{align*}
\Gamma & \vdash c : \Delta, \alpha : A
\end{align*}
\]

\[
\begin{align*}
(x \notin \text{dom}(\Gamma), \alpha \notin \text{dom}(\Delta))
\end{align*}
\]

Interchange:

\[
\begin{align*}
\Gamma, x : A, y : B, \Gamma' & \vdash t : C | \Delta \\
\Gamma, y : B, x : A, \Gamma' & \vdash t : C | \Delta \\
\Gamma, x : A, y : B, \Gamma' & \vdash k : C | \Delta \\
\Gamma, y : B, x : A, \Gamma' & \vdash k : C | \Delta \\
\Gamma, x : A, y : B, \Gamma' & \vdash c : \Delta \\
\Gamma, y : B, x : A, \Gamma' & \vdash c : \Delta \\
\Gamma, x : A, y : B, \Gamma' & \vdash c : \Delta \\
\Gamma, y : B, x : A, \Gamma' & \vdash c : \Delta \\
\end{align*}
\]

Contraction:

\[
\begin{align*}
x : A, y : A, \Gamma & \vdash t : A | \Delta \\
\Gamma, x : A, \Gamma & \vdash t[x/y] : A | \Delta \\
\Gamma, x : A, \Gamma & \vdash k[x/y] : A | \Delta \\
\Gamma, x : A, \Gamma & \vdash c[x/y] : A | \Delta \\
\end{align*}
\]

\[
\begin{align*}
\Gamma & \vdash t : A | \Delta, \alpha : A, \beta : B, \Delta' \\
\Gamma & \vdash t : A | \Delta, \beta : B, \alpha : A, \Delta' \\
\Gamma & \vdash k : C | \Delta, \alpha : A, \beta : B, \Delta' \\
\Gamma & \vdash k : C | \Delta, \beta : B, \alpha : A, \Delta' \\
\Gamma & \vdash c : \Delta, \alpha : A, \beta : B, \Delta' \\
\Gamma & \vdash c : \Delta, \beta : B, \alpha : A, \Delta' \\
\end{align*}
\]

Cut:

\[
\begin{align*}
\Gamma & \vdash t : A | \Theta \\
\Delta & \vdash k : A | \Lambda \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, \Delta & \vdash t \cdot k : \Theta, \Lambda
\end{align*}
\]

\[
\begin{align*}
(\text{dom}(\Gamma) \cap \text{dom}(\Delta) = \emptyset = \text{dom}(\Theta) \cap \text{dom}(\Lambda))
\end{align*}
\]

Table 2.6: Structural typing rules of the Dual Calculus
Disjunction:

\[ \Gamma ⊢ t : A | \Delta \quad \text{(RV}_1\text{)} \]
\[ \Gamma ⊢ t_1(t) : A \lor B | \Delta \quad \text{(RV}_2\text{)} \]
\[ \Gamma | k : A \vdash \Delta \quad \Gamma | k' : B \vdash \Delta \]
\[ (L\lor) \]
\[ \Gamma | [k, k'] : A \lor B \vdash \Delta \]

Conjunction:

\[ \Gamma ⊢ t : A | \Delta \quad \Gamma ⊢ t' : B | \Delta \quad \text{(R\land)} \]
\[ \Gamma ⊢ \langle t, t' \rangle : A \land B | \Delta \quad \text{(
\text{R} \land_2\text{)} } \]
\[ \Gamma | k : A \vdash \Delta \quad \Gamma | k : B \vdash \Delta \]
\[ \text{(
\text{L} \land_1\text{)} } \]
\[ \Gamma | fst[k] : A \land B \vdash \Delta \quad \Gamma | snd[k] : A \land B \vdash \Delta \]

Negation:

\[ \Gamma | k : A \vdash \Delta \quad \text{(R\neg)} \]
\[ \Gamma ⊢ not(k) : \neg A | \Delta \quad \text{(
\text{R} \neg_1\text{)} } \]
\[ \Gamma | t : A | \Delta \quad \text{(
\text{L} \neg_1\text{)} } \]
\[ \Gamma | not[t] : \neg A \vdash \Delta \]

Table 2.7: Operational typing rules for the Dual Calculus

\[ \Gamma ⊢ c \vdash \Delta, x : A \quad \Gamma ⊢ c \vdash \Delta \]
\[ \Gamma ⊢ \alpha.(c) : A | \Delta \quad \Gamma | x.(c) : A \vdash \Delta \]

Table 2.8: Typing rules for abstractions in the Dual Calculus

31
2.3 Proving and Programming with DC

§2.30. Programming by Proving The calculus duly presented, let’s see how the arsenal of DC can be used to write interesting programs. We leave the dynamics of the language for the next section and so cannot yet justify interestingness from a semantic point-of-view. However, the theorems of LK, via the Curry-Howard correspondence that we aim to explore, provide an alternative source of interest. We begin with a few familiar constructions, followed by the precise statement of the logical content of DC phrases (theorem 2) and then give, precisely and generally, an account of how derivations in LK induce programs in DC (theorem 3). Finally, we also explore another avenue for getting programs from logical theorems using the very duality of the calculus.

§2.31. Example: Double-negation Elimination As we don’t have, yet, implication in DC, we prove double-negation elimination ($\neg\neg A \rightarrow A$) using entailment: we take an assumption $x : \neg\neg A$ and to construct a term which has type $A$:

$$x : \neg\neg A \vdash ? : A.$$  

The operational rules all have in common that they build more complex formulas than what they are given—so, for arbitrary $A$, they cannot be the last rule of the derivation of the type. Identity, thinning, and contraction offer no help—and the cut rule types cuts, not terms. Therefore the outermost term must be a co-abstraction—on, say, $\alpha$. The proof obligation is then to find a cut using $x : \neg\neg A$ and $\alpha : A$.

$$x : \neg\neg A \vdash ? \vdash \alpha : A$$

Now, negating $\alpha$ twice leads to the co-term $\not\not[\not\not(\alpha)]$ of (obviously) type $\neg\neg A$, which can be cut with $x$. Putting it all together we have:

$$| \alpha : A \vdash \alpha : A$$

$$\vdash \not\not(\alpha) : \neg A \vdash A$$

$$x : \neg\neg A \vdash x : \neg\neg A$$

$$| \not\not[\not\not(\alpha)] : \neg\neg A \vdash \alpha : A$$

$$x : \neg\neg A \vdash x \cdot \not\not[\not\not(\alpha)] \vdash \alpha : A$$

$$x : \neg\neg A \vdash \alpha.(x \cdot \not\not[\not\not(\alpha)]) : A$$

§2.32. Continuations Are Logical Too The above derivation looks far more complicated than the earlier derivation of double-negation elimination in LK (2.1, p. 23). That (two-step) derivation ends in a left rule and, we intuited, left rules correspond to co-terms. Co-terms (or continuations) have the same logical basis as terms. With this in mind, we see that the above term hid within itself a simpler (co-term) derivation of double-negation elimination:

$$| \alpha : A \vdash \alpha : A$$

$$\vdash \not\not(\alpha) : \neg A \vdash A$$

$$| \not\not[\not\not(\alpha)] : \neg\neg A \vdash \alpha : A$$
§2.33. Example: Law of the Excluded Middle  The Law of the Excluded Middle offers a slightly more involved challenge but it also shows the computational power of the calculus. This time we start with no assumptions. The outermost operation cannot be an injection: this would imply that we could prove either every formula \( A \) or the negation of every formula, \( \neg A \). We must start with a co-abstraction. This gives us a co-variable \( \alpha \) of type \( \neg A \lor A \) which will contain the current calling context. From here there is a Game Semantics interpretation of the excluded middle that we can use to guide our search\(^8\): We proceed execution as a term of type \( \neg A \lor A \). Should this term ever be used, it must be because it was passed a co-term of type \( \neg A \lor A \), i.e. a term of type \( A \) encapsulated as a continuation. So we do the opposite and create a term that is the encapsulation of a continuation that takes a value of type \( A \) and redoes the entire computation with this value. We can backtrack like this because we capture the original continuation in \( \alpha \) at the start of the execution:

\[
\alpha.(i_1\langle \text{not}(x.(i_2(x) \cdot \alpha)) \rangle \cdot \alpha);
\]

we can, then, prove that it has the appropriate type\(^9\):

\[
\begin{align*}
  x : A & \vdash x : A \mid \\
  x : A & \vdash i_2(x) : \neg A \lor A \mid \\
  x : A & \vdash i_2(x) \cdot \alpha \vdash \alpha : \neg A \lor A \\
  x \cdot (i_2(x) \cdot \alpha) & : A \vdash \alpha : \neg A \lor A \\
  \vdash \text{not}(x.(i_2(x) \cdot \alpha)) : \neg A \lor A \mid \\
  \vdash i_1\langle \text{not}(x.(i_2(x) \cdot \alpha)) \rangle \cdot \alpha' \vdash \alpha : \neg A \lor A, \alpha' : \neg A \lor A \\
  \vdash i_1\langle \text{not}(x.(i_2(x) \cdot \alpha)) \rangle \cdot \alpha' \vdash \alpha : \neg A \lor A \\
  \vdash \alpha.(i_1\langle \text{not}(x.(i_2(x) \cdot \alpha)) \rangle \cdot \alpha) : \neg A \lor A \mid
\end{align*}
\]

§2.34. Example: De Morgan’s Laws  The final example we will tackle in this section is the de Morgan Law \( \neg(A \land B) \rightarrow \neg A \lor \neg B \). We chose this one as it is the only truly Classical of the de Morgan Laws. For variety, instead of finding a term, we shall focus on finding a continuation of type \( \neg(A \land B) \) under a context \( \alpha : \neg A \lor \neg B \). From now on we shall omit mentions to (obvious) thinnings, contractions, and interchanges from typing derivations.

If we start by using the operational rules for \( \neg \) and \( \land \) we find ourselves needing terms of type \( A \) and \( B \) under the hypothesis \( \alpha \)—and no obvious operational rule to proceed with. Using a co-abstraction gives us one more assumption co-variable (say) \( \beta : A \) to play with. But now we can turn \( \beta \) into a term \( \text{not}(\beta) : \neg A \) which we can pass on to \( \alpha \)

---

\(^8\)Wadler [2003, section 4, p. 195] offers a more entertaining explanation of this term.

\(^9\)NB: The use of contraction in the penultimate step (cf. §3.11).
using the first injection. In full we have:

\[
\begin{align*}
| \beta : A \vdash \beta : A & \quad \vdash \text{not}(\beta) : \neg A \mid \beta : A \\
\vdash t_1(\text{not}(\beta)) : \neg A \lor \neg B & \quad | \alpha : \neg A \lor \neg B \vdash \alpha : \neg A \lor \neg B \\
\vdash t_1(\text{not}(\beta)) \cdot \alpha \vdash \alpha : \neg A \lor \neg B, \beta : A & \quad \vdash \beta.(t_1(\text{not}(\beta)) \cdot \alpha) : A \mid \alpha : \neg A \lor \neg B \\
\vdash (\beta.(t_1(\text{not}(\beta)) \cdot \alpha), \beta.(t_2(\text{not}(\beta)) \cdot \alpha)) : A \land B & \quad | \alpha : \neg A \lor \neg B
\end{align*}
\]

(same for \(t_2(\ldots), \beta : B\))

§2.35. DC as a Proof System  Let us follow down this path, by formalizing the connection between proofs in LK and programs in DC. Following in the spirit of the Curry-Howard correspondence—and as we have done so far—we confuse types and propositions. Given a DC variable typing context \(\Gamma = x_1 : A_1, \ldots, x_n : A_n\), its LK counterpart is defined as \(\|\Gamma\| = A_1, \ldots, A_n\); for contexts of co-variables \(\Delta = \alpha_m : A_m, \ldots, \alpha_1 : A_1\), we have \(\|\Delta\| = A_m, \ldots, A_1\). Further, we can define a translation between DC judgements and LK sequents:

\[
\begin{align*}
\| \Gamma \vdash t : A \mid \Delta \| & \equiv \| \Gamma \| \vdash \| \Delta \|, A \\
\| \Gamma \mid k : A \vdash \Delta \| & \equiv \| A, \| \Gamma \| \vdash \| \Delta \| \\
\| \Gamma \vdash c : \Delta \mid \| & \equiv \| \Gamma \| \vdash \| \Delta \|
\end{align*}
\]

Note that the type of a (co-) term is positioned to the left (right) of the (co-) context. This is to keep the connection with the LK left and right rules which apply only to the outermost formulas.

**Theorem 2.** Typings in DC induce theorems in LK:

\[
\begin{align*}
\Gamma \vdash t : A \mid \Delta & \implies \| \Gamma \vdash t : A \mid \Delta \| \\
\Gamma \mid k : A \vdash \Delta & \implies \| \Gamma \mid k : A \vdash \Delta \| \\
\Gamma \vdash c : \Delta & \implies \| \Gamma \vdash c : \Delta \|
\end{align*}
\]

**Proof.** The proof is by mutual induction on the typing rules. Most DC operators have direct LK counterparts; abstractions, despite being the exception to this rule, are quite simple to handle:

\[
\begin{align*}
\frac{(\Gamma \vdash c : \Delta, \alpha : A)}{\| \Gamma \vdash c : \Delta, \alpha : A \|} & \quad \text{IH} \\
\frac{(x : A, \Gamma \vdash c : \Delta)}{\| x : A, \Gamma \vdash c : \Delta \|} & \quad \text{IH} \\
\frac{\| \Gamma \| \vdash \| \Delta \|, A}{\| \| \Gamma \| \vdash \| \Delta \|, A} & \quad \text{IH} \\
\frac{\| \Gamma \mid a.(c) : A \mid \Delta \|}{\| \Gamma \| \mid x.(c) : A \vdash \Delta \|} & \quad \text{IH}
\end{align*}
\]

\(\square\)
§2.36. Closed Cuts  A curious consequence of this theorem and of the Hauptsatz—and which serves as a sanity check of sorts—is that there is no cut without free variables and without free co-variables. Everything must either consume and/or create something.

**Corollary 1.** There is no well-typed closed DC cut \( \vdash c \vdash \). This corollary is enough to disqualify the closed cut \( \alpha : \not \langle \alpha \rangle \cdot \alpha / \not \langle \alpha \rangle \cdot \alpha \) from being well-typed. This cut functions somewhat like the infamous self application term of the lambda calculus \( \omega \equiv (\lambda x. x) (\lambda x. x) \) in that, given the separation of terms and co-terms, it is the closest one can get to a closed cut of a term with itself. Like \( \omega \), this term is non-terminating in our system (cf. reduction rules in section 2.4).

§2.37. From LK to DC  There is also a converse to theorem 2. We must begin by defining mappings from LK contexts to DC contexts. For this we need to assign names to the different assumptions (or consequences). For simplicity, we will assume the existence of distinguished indexed sets \( \{ x_n \}_{n \in \mathbb{N}} \) and \( \{ \alpha_n \}_{n \in \mathbb{N}} \) of variables and co-variables. With these sets, and for sequents \( A_n, \ldots, A_1 \vdash B_1, \ldots, B_m \), we define the mappings:

\[
\begin{align*}
[A_n, \ldots, A_1]^\ast &= x_n : A_n, \ldots, x_1 : A_1 \\
[B_1, \ldots, B_m]^\ast &= \alpha_1 : B_1, \ldots, \alpha_m : B_m;
\end{align*}
\]

and, more generally, for \( k \in \mathbb{N} \)

\[
\begin{align*}
[A_n, \ldots, A_1]^\ast_k &= x_{k+n} : A_n, \ldots, x_{k+1} : A_1 \\
[B_1, \ldots, B_m]^\ast_k &= \alpha_{k+1} : B_1, \ldots, \alpha_{k+m} : B_m;
\end{align*}
\]

**Theorem 3.** If we can derive \( \Gamma \vdash \Delta \) in LK then for some cut \( c \) of DC

\[
[\Gamma]^\ast \vdash c \vdash [\Delta]^\ast.
\]

**Proof.** We prove the more general statement that for all \( k, l \in \mathbb{N} \) we have

\[
\Gamma \vdash \Delta \implies [\Gamma]^\ast_k \vdash c \vdash [\Delta]^\ast_l
\]

for some cut \( c \). We omit the laborious derivations, but remark that we need to assume that typings are preserved by transposition of (co-) variable names.

§2.38. Term/Co-term Duality  It is easy to extend the duality of LK to DC. Terms and co-terms are dual, whereas the dual of a cut is still a cut. The complicated issue is that, in translating phrases, variables become co-variables, and vice-versa. To simplify things, we assume a bijection \((-)^\ast\) between variables and co-variables—by abuse of notation we represent the inverse of this bijection also by \((-)^\ast\). With it we can define the duality of DC as in tables 2.9 and 2.10. We claim without formal proof that this duality preserves typings:
THE DUAL CALCULUS

\[ \begin{align*}
X^\circ &= X \\
(A \land B)^\circ &= B^\circ \lor A^\circ \\
(A \lor B)^\circ &= B^\circ \land A^\circ \\
(\neg A)^\circ &= \neg A^\circ \\
(x_1 : A_1, \ldots, x_n : A_n)^\circ &= x_n^\circ : A_n^\circ, \ldots, x_1^\circ : A_1^\circ \\
(a_n : A_n, \ldots, a_1 : A_1)^\circ &= a_1^\circ : A_1^\circ, \ldots, a_n^\circ : A_n^\circ \\
\end{align*} \]

Table 2.9: Duality of DC types and contexts.

\[ \begin{align*}
\langle t_1, t_2 \rangle^\circ &= [t_2^\circ, t_1^\circ] \\
[k_1, k_2]^\circ &= \langle k_2^\circ, k_1^\circ \rangle \\
t_1(t)^\circ &= \text{fst}[t^\circ] \\
t_2(t)^\circ &= \text{snd}[t^\circ] \\
\text{not}(k)^\circ &= \text{not}[k^\circ] \\
\alpha.(c)^\circ &= \alpha^\circ.(c^\circ) \\
\end{align*} \]

\[ (t \cdot k)^\circ = k^\circ \cdot t^\circ \]

Table 2.10: Duality for the syntax of DC.
REDUCTION

Proposition 4.

\[ \Gamma \vdash t : A \mid \Delta \Rightarrow \Delta^o \mid \tau^o : A^o \dashv \Gamma^o \]

\[ \Gamma \mid k : A \vdash \Delta \Rightarrow \Gamma^o \vdash k^o : A^o \mid \Gamma^o \]

\[ \Gamma \vdash c \vdash \Delta \Rightarrow \Delta^o \vdash c^o \vdash \Gamma^o \]

This proposition suggests a new way of devising proofs/programs. Let’s go back to the Classical de Morgan Law, \( \neg (A \land B) \rightarrow \neg A \lor \neg B \). To find a term that witnesses this implication, we proceed by finding a co-term that witnesses the de Morgan Law \( \neg B^o \land \neg A^o \rightarrow \neg (B^o \lor A^o) \) and dualize it. We, thus, reduce the problem of finding a Classical proof to that of finding an Intuitionistic one—hopefully a simpler one.

Indeed there is one such co-term if we take a co-variable \( \alpha \) of type \( \neg (B^o \lor A^o) \):

\[ p.(p \cdot fst[nb.(p \cdot snd[na.not[[x.(nb \cdot not[x]), x.(na \cdot not[x])]} \cdot \alpha)])] \]

by dualization we obtain (for \( x : \neg (A \land B) \)):

\[ \gamma.(\alpha.(\iota_1(\beta.(x \cdot not[[\delta.(not(\delta) \cdot \alpha), \delta.(not(\delta) \cdot \beta)]}) \cdot \gamma)) \cdot \gamma) \]

which does, in fact, have the right type:

\[ \vdash not(\delta) : \neg A \mid \delta : A \mid \alpha : \neg A \vdash \alpha : \neg A \]

\[ \vdash not(\delta) \cdot \alpha \vdash \alpha : \neg A, \delta : A \]

\[ \vdash \delta.(not(\delta) \cdot \alpha) : A \mid \alpha : \neg A \quad \text{likewise for } \beta : \neg B \]

\[ \vdash \langle \delta.(not(\delta) \cdot \alpha), \delta.(not(\delta) \cdot \beta) \rangle : A \land B \mid \alpha : \neg A, \beta : \neg B \]

\[ \vdash \langle \delta.(not(\delta) \cdot \alpha), \delta.(not(\delta) \cdot \beta) \rangle : \neg (A \land B) \quad \vdash \alpha : \neg A, \beta : \neg B \]

\[ x : \neg (A \land B) \vdash x \cdot not[[\ldots, \ldots]] \vdash \alpha : \neg A, \beta : \neg B \]

\[ x : \neg (A \land B) \vdash \beta.(x \cdot not[[\ldots, \ldots]]) \vdash \neg B \mid \alpha : \neg A \]

\[ x : \neg (A \land B) \vdash \iota_2(\beta.(x \cdot not[[\ldots, \ldots]]) \cdot \gamma) \vdash \neg A \lor \neg B \mid \gamma : \neg A \lor \neg B \]

\[ x : \neg (A \land B) \vdash \gamma.(\iota_1(\alpha.\iota_2(\beta.(x \cdot not[[\ldots, \ldots]]) \cdot \gamma)) \cdot \gamma) \vdash \neg A \lor \neg B \]

2.4 Reduction

§2.39. Cutting Cuts. Cuts combine computations (terms) with continuations (co-terms). Reduction in DC is the execution of this coupling. So, for example, when we cut a pair with a continuation representing a projection,

\[ \langle t, u \rangle \cdot \text{fst}[k]. \]
§2.40. Reducing Abstractions  The two types of abstractions have the same underlying reduction behaviour: they capture whatever is passed to them and use that to replace the abstracted (co-)variable in the body of the abstraction—yielding, importantly, a new cut. The formal rules can be found in table 2.13.

§2.41. Lafont’s Critical Pair  Alas, the reduction rules for abstractions are quite misbehaved. Notice that the co-abstraction rule accepts an arbitrary continuation—in particular it accepts abstractions; similarly, the abstraction rule accepts arbitrary co-abstractions for the term component. Consequently, given a cut of the form \( \alpha.(c) \cdot x.(d) \) we are left with the following general situation

\[
\alpha.(c) \cdot x.(d)
\]

\[
c[x.(d)/a] \quad \quad d[\alpha.(c)/x]
\]

and we have no reason to believe the two paths should ever converge. This observation has been attributed to Lafont (Girard et al., 1989, pp. 151–152) who, furthermore,

\[
\alpha.(c) \cdot k \ Published in 1989.}
EXTENSIONS OF THE DUAL CALCULUS

\[ p := x \mid \langle t, t' \rangle \mid i_1(t) \mid i_2(t) \mid \text{not}(k) \]

\[ \alpha.(c) \cdot k \rightarrow c[k/\alpha] \]

\[ p \cdot x.(c) \rightarrow c[p/x] \]

Table 2.14: The co-abstraction first subsystem of DC

noticed the following—rather unfortunate—corollary:

Take two arbitrary, well-typed, cuts \( \Gamma \vdash c \dashv \Delta \) and \( \Gamma \vdash d \dashv \Delta \), and assumed not to have any free variables or co-variables in common. Additionally, take \( x \) and \( \alpha \) fresh, for both. Then, the following happens:

\[ \alpha.(c) \cdot x.(d) \]

\[ \begin{array}{c}
\vdash c[x.(d)/\alpha] \\
\vdash [\alpha.(c)/x]
\end{array} \]

\[ \parallel \parallel \]

\[ c \]

\[ d \]

This is a problem because if one wishes to equate cuts by reduction—in the same manner that we equate \( 2 + 2 \) and \( 4 \)—then all cuts become unified. More precisely, any sound interpretation of DC in which equality respects the transitive symmetric closure of reduction is, essentially, trivial:

\[ c \leftrightarrow \alpha.(c) \cdot x.(d) \rightarrow d \]

\[ \Downarrow \quad \Downarrow \quad \Downarrow \quad \text{(closure)} \]

\[ c = \alpha.(c) \cdot x.(d) = d \]

\[ \Downarrow \quad \Downarrow \quad \Downarrow \quad \text{(soundness)} \]

\[ [c] = [\alpha.(c) \cdot x.(d)] = [d] \]

§2.42. Abstraction and Co-abstraction Prioritization We present in tables 2.14 and 2.15 two subsystems of DC for which (head) reduction is deterministic and, therefore, confluent. The obvious thing to do if we wish to generate a confluent system based on DC is to give priority to one of the kinds of abstraction over the other. Note that the operational reduction rules remain the same, only the abstraction reduction rules change—hence, this system can be generalized to other connectives in the obvious manner.

2.5 Extensions of the Dual Calculus

§2.43. Beyond the Dual Fragment We have now surveyed the dual fragment of the Dual Calculus as it was originally presented (Wadler, 2003). This fragment was part of a larger system that also included implication but not its dual, subtraction (Crolard, 2004; Curien and Herbelin, 2000), nor did it include second-order types (Tzevelekos, 2004).
THE DUAL CALCULUS

\[ l := a \mid \text{fst}[k] \mid \text{snd}[k] \mid [k, k'] \mid \text{not}[t] \]

\[ t \cdot x.(c) \rightsquigarrow c[t/x] \]

\[ a.(c) \cdot l \rightsquigarrow c[l/a] \]

Table 2.15: The abstraction first subsystem of DC

We will now extend the system with both implication and subtraction, thus retaining self-duality of the calculus (tables 2.16 and 2.17). This is not, however, the simple reason why we include subtraction: subtraction will be an indispensable part of our definition of Mendler induction; dually, implication will appear in the definition of co-induction (chapter 5). Finally, we will also discuss the extension to second-order types—universal and existential (tables 2.18 and 2.19).

§2.44. Implication  A term of type \( A \rightarrow B \) is a term of type \( B \) that can be instantiated by terms of type \( A \). Like in the lambda-calculus, the values of this type are those lambda-abstractions \( \lambda x.(t) \) where \( x \) (assumed of type \( A \)) may appear free in \( t \). The left operation—the associated co-term constructor—indicates application to a specific term \( t' \). It is an eliminator (as all co-terms are) because when we cut it with a lambda-abstraction of type \( A \rightarrow B \) it yields an instance which has type \( B \). However, this instantiation yields terms whereas cuts reduce to cuts; the application co-term, then, needs also a continuation \( k \) for the result of the instantiation. We represent application by \( (t' @ k) \). The continuation co-term must have type \( B \) (the type of the instance) for the application to be well-typed.

§2.45. Subtraction  We obtain subtractive types and its operations by essentially “dualizing everything”: if an implication \( A \rightarrow B \) abstracts terms of type \( B \), a subtraction \( A - B \) abstracts co-terms of type \( A \); if lambda-abstractions act on variables, subtractive, or mu-, abstractions \( \mu a.(k) \) (which we also call catches) act on co-variables; if eliminations for implications are pairings of the instance term with a continuation, the introductions for subtractions are the pairing of a co-term \( k' \) with an initial value term \( t \) as \( (t \# k') \). Subtractive cuts \( (t \# k') \cdot \mu a.(k) \) reduce to the composition of the continuations \( t \cdot k[k'/\alpha] \). It is this compositional ability that will be explored for Mendler induction later on.

§2.46. Quantification  To complete the picture, vis-a-vis \( LK \), we can add universal and existential quantification to DC, in the manner of [Kimura and Tatsuta, 2009]. The syn-

[10] In their detailed study of the applications of \( LK \)-based calculi to the study of reduction orders of the lambda-calculus, [Curien and Herbelin, 2000] offered an alternative rule for implication:

\[ \lambda x.(t) \cdot (t' @ k) \rightsquigarrow t' \cdot x.(t \cdot k) \quad (x \notin \text{fv}(k)). \]

The advantage of this rule is that simply by fixing the underlying reduction to be abstraction or co-abstraction prioritizing, one can directly interpret the call-by-name and call-by-value variants of the lambda-calculus. In the interest of historical accuracy, we remark that [Wadler, 2003] also uses this rule.
CURRY-HOWARD CORRESPONDENCE OF LK AND THE DUAL CALCULUS

\[ t := \ldots \mid \lambda x.(t) \mid (t \# k) \]
\[ k := \ldots \mid (t @ k) \mid \mu \alpha.(k) \]

\[ x : A, \Gamma \vdash t : B \mid \Delta \]
\[ \Gamma \vdash \lambda x.(t) : A \rightarrow B \mid \Delta \]
\[ \Gamma \vdash t : A \mid \Delta \]
\[ \Gamma \vdash (t @ k) : A \rightarrow B \vdash \Delta \]

\[ \Gamma \vdash t : A \mid \Delta \]
\[ \Gamma \vdash k : B \vdash \Delta \]
\[ \Gamma \vdash \lambda x.(t)(\# k) : A - B \mid \Delta \]
\[ \Gamma \vdash (t \# k)(\mu \alpha.(k)) \rightarrow t \bullet k[\alpha / k'] \]

Table 2.16: Terms and typings of the implicative fragment of DC

Table 2.17: Reduction rules for the implicative fragment of DC

tax (see table 2.18) is now extended with four operators corresponding to each of the left and right-rules of each quantifier. The typing rules (also in table 2.18) mimic the corresponding LK rules—including, rather fundamentally, the restriction over the quantified type variable, that it does not appear free in the types present in the judgement.

2.6 Curry-Howard Correspondence of LK and the Dual Calculus

§2.47. Terms Are Not Right-rules, etc. As a way to conclude this chapter, let us discuss the status of the Dual Calculus as the Curry-Howard equivalent of Gentzen’s LK. The term forming rules of DC are not remotely the same as LK’s right-rules: the former expect other specific types of phrases to precede them; the latter can be used regardless of which rule ends the previous derivation. Similarly, co-terms and cuts expect very specific types of phrases at specific positions, whereas their LK counterparts do not. Indeed, cuts in LK can follow two right-rules, or two-left rules, or two cuts, or any other combination of two rules of LK. Tellingly, the theorems which state the provability correspondence between DC and LK (theorems 2 and 3) are stated or proven in terms of cuts. Lacking the phrase agnosticism of LK, DC needs the two types of abstraction to coerce cuts into (co-) terms (and application to (co-) variables for the converse). And this coercion means that the derivation rules in LK and DC, though related, do not have the exact same structure.
THE DUAL CALCULUS

\[ t := \ldots | a(t) | e(t) \]
\[ k := \ldots | a[k] | e[k] \]

\[
\begin{align*}
\Gamma^\ast \vdash t : F(X) | \Delta^\ast & \quad \Gamma | k : F(T) \vdash \Delta \\
\Gamma^\ast \vdash a(t) : \forall X . F(X) | \Delta^\ast & \quad \Gamma | a[k] : \forall X . F(X) \vdash \Delta \\
\Gamma \vdash t : F(T) | \Delta & \quad \Gamma^\ast | k : F(X) \vdash \Delta^\ast \\
\Gamma \vdash e(t) : \exists X . F(X) | \Delta & \quad \Gamma^\ast | e[k] : \exists X . F(X) \vdash \Delta^\ast
\end{align*}
\]

(for \( X \) not free in the contexts \( \Gamma^\ast, \Delta^\ast \).

Table 2.18: Second-order terms and typings

\[ a(t) \cdot a[k] \Rightarrow t \cdot k \quad e(t) \cdot e[k] \Rightarrow t \cdot k \]

Table 2.19: Reduction for second-order cuts

Compare this situation to the simply typed lambda-calculus where each term determines its typing derivation exactly and, in turn, also a derivation in intuitionistic logic.

§2.48. Cuts Are Not Cuts In terms of the dynamics of the language, it also becomes clear from the rules above that what in DC we call a ‘cut’ does not function analogously to a Gentzian ‘cut’: The process of reduction turns a cut into another cut—so no hope for a cut elimination result! This problem was noticed by Curien and Herbelin (2000); and to fix it, one must disentangle what we call syntactically a cut, from the cut typing rule. In the context of the Dual Calculus, Wadler (2003) offers an alternative formulation of the typing system that splits cuts into four cases: cuts without co- and variables which are typed using the cut rule, cuts without variables, cuts without co-variables, and the identity cut. Only cuts of the first kind are typed using the cut rule; the remainder have their own typing rules. The downside of this approach is that the typing systems stops being directed by syntax, unless we add more operators to accommodate the new cases. But more, some of the extra typing derivation do not have corresponding (primitive) rules in \( LK \). Perhaps this is why Wadler did not ultimately use this as DC’s typing system. In any case, it feels clear to us that, even if intimately related, DC and \( LK \) should not be treated as Curry-Howard equivalents.

§2.49. The Beauty of DC That being said, DC retains one of \( LK \)’s best features: its structural regularity. The duality remains and is, indeed, refined with the syntactical restriction on the rules. Starting from this simple system, and by introducing fairly intuitive
restrictions—call-by-name and call-by-value—we end up with systems strongly related to
languages with control (Wadler, 2005) and to monadic meta-languages (Wadler, 2003).

§2.50. DC a Better LK? Proof theory post-dates Gentzen. His interest in provability
boiled down to finding true propositions in either the Intuitionistic or Classical sense.
In particular, it lacks a good notion of equivalence of proofs—whose importance only
crystallized after Gentzen’s untimely death. It is exactly this that DC’s subsystems (seen
as a proof systems) seem to encompass so elegantly.
THE DUAL CALCULUS

Phrases

\[ t := x \in \text{Var} \mid \langle t, t \rangle \mid t_1(t) \mid t_2(t) \mid \neg t \mid \lambda x.(t) \mid (t \# k) \mid a(t) \mid e(t) \mid \alpha(c) \]
\[ k := \alpha \in \text{Covar} \mid fst[k] \mid snd[k] \mid [k, k] \mid \neg t \mid (t @ k) \mid \mu x.(k) \mid a[k] \mid e[k] \mid \alpha(c) \]
\[ c := t \cdot k \]

Typing Rules

Identity:

\[
\begin{array}{c}
x : A \vdash x : A \\
\end{array} \quad \begin{array}{c}
\alpha : A \vdash \alpha : A \\
\end{array}
\]

(Weakening, Contraction and Interchange omitted)

Disjunction:

\[
\begin{array}{c}
\Gamma \vdash t : A | \Delta \\
\Gamma \vdash \iota_1(t) : A \lor B | \Delta \\
\Gamma \vdash \iota_2(t) : A \lor B | \Delta \\
\end{array} \quad \quad \begin{array}{c}
\Gamma \vdash t : B | \Delta \\
\Gamma \vdash \iota_1(t) : A \lor B | \Delta \\
\Gamma \vdash \iota_2(t) : A \lor B | \Delta \\
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash k : A \rightarrow \Delta \\
\Gamma \vdash k' : B \rightarrow \Delta \\
\Gamma \mid [k, k'] : A \lor B \rightarrow \Delta
\end{array}
\]

Conjunction:

\[
\begin{array}{c}
\Gamma \vdash t : A | \Delta \\
\Gamma \vdash t' : B | \Delta \\
\end{array} \quad \begin{array}{c}
\Gamma \vdash \langle t, t' \rangle : A \land B | \Delta \\
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash k : A \rightarrow \Delta \\
\Gamma \mid fst[k] : A \land B \rightarrow \Delta
\end{array} \quad \begin{array}{c}
\Gamma \vdash k : B \rightarrow \Delta \\
\Gamma \mid snd[k] : A \land B \rightarrow \Delta
\end{array}
\]

Negation:

\[
\begin{array}{c}
\Gamma \mid k : A \rightarrow \Delta \\
\Gamma \vdash \neg \langle k \rangle : \neg A | \Delta \\
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash t : A | \Delta \\
\Gamma \vdash not[t] : \neg A \rightarrow \Delta
\end{array}
\]

Implication:

\[
\begin{array}{c}
x : A, \Gamma \vdash t : B | \Delta \\
\Gamma \vdash \lambda x.(t) : A \rightarrow B | \Delta \\
\end{array} \quad \begin{array}{c}
\Gamma \vdash t : A | \Delta \\
\Gamma \vdash (t @ k) : A \rightarrow B \rightarrow \Delta
\end{array}
\]

Table 2.20: Summary of the typing and reduction rules of the second-order Dual Calculus (I of II)
Typing Rules (Continued)

**Subtraction:**
\[
\begin{array}{c}
\Gamma \vdash t : A | \Delta \\
\Gamma \vdash k : B \vdash \Delta \\
\end{array}
\]
\[
\Gamma \vdash (t \# k) : A - B | \Delta
\]
\[
\Gamma \vdash k : A \vdash \Delta, a : B
\]
\[
\Gamma \vdash \mu a.(k) : A - B \vdash \Delta
\]

**Universal Quantification:**
\[
\Gamma^\ast \vdash t : F(X) | \Delta^\ast
\]
\[
\Gamma^* \vdash a(t) : \forall X . F(X) | \Delta^* \]
\[
\Gamma \vdash k : F(T) \vdash \Delta
\]
\[
\Gamma \vdash a[k] : \forall X . F(X) \vdash \Delta
\]

**Existential Quantification:**
\[
\Gamma \vdash t : F(T) | \Delta
\]
\[
\Gamma \vdash e(t) : \exists X . F(X) | \Delta
\]
\[
\Gamma^* \vdash k : F(X) \vdash \Delta^* \]
\[
\Gamma^* \vdash e[k] : \exists X . F(X) \vdash \Delta^*
\]

(for \( X \) not free in the contexts \( \Gamma^*, \Delta^* \)).

**Abstractions:**
\[
\Gamma \vdash c \vdash \Delta \quad \alpha : A
\]
\[
\Gamma \vdash \alpha .(c) : A | \Delta
\]
\[
\Gamma \vdash x : A, \Gamma \vdash c \vdash \Delta
\]
\[
\Gamma \vdash x.(c) : A \vdash \Delta
\]

**Cut:**
\[
\Gamma \vdash t : A | \Theta \\
\Delta \vdash k : A \vdash \Lambda
\]
\[
\Gamma, \Delta \vdash t \bullet k \vdash \Theta, \Lambda
\]

**Reduction Rules**
\[
\langle t, t' \rangle \bullet fst[k] \leftrightarrow t \bullet k
\]
\[
\langle t, t' \rangle \bullet snd[k] \leftrightarrow t' \bullet k
\]
\[
\text{not}(k) \bullet \text{not}[t] \leftrightarrow t \bullet k
\]
\[
\lambda x . t \bullet (t' \circ k) \rightarrow t'[x / t] \bullet k
\]
\[
(x \# k') \bullet \mu a.(k) \rightarrow t \bullet k[k'/\alpha]
\]
\[
a(t) \bullet a[k] \rightarrow t \bullet k
\]
\[
e(t) \bullet e[k] \rightarrow t \bullet k
\]
\[
\alpha .(c) \bullet k \leftrightarrow c[k/\alpha]
\]
\[
t \bullet x .(c) \leftrightarrow c[t/x]
\]

Table 2.21: Summary of the typing and reduction rules of the second-order Dual Calculus (II of II)
THE DUAL CALCULUS
Chapter 3
Reduction Properties of the Dual Calculus

§3.1. In Search of the Greatest Generality  We dedicate this chapter to the analysis of reduction in the Dual Calculus. Specifically, we shall engage in proving four properties: duality, substitutivity, preservation, and—our main topic—strong normalization. We shall aim for the utmost generality, encompassing the behaviour of different reduction strategies and of different types in one analysis. The exact calculus we will deal with will be the base one extended with implicational and subtractive, and second-order types; and the reduction will be the non-confluent one where abstractions and co-abstractions have equal reduction precedence. Further, we will consider not just head reduction (sections 2.4 and 2.5) but full parallel reduction, where inner-cuts can also reduce. For this extra effort we will reap rewards in the form of proofs of the strong normalization and preservation properties of any sub-calculus where the reduction rules are a subset of these—in particular, the deterministic (co-) abstraction-prioritizing head reduction.

§3.2. Structure of the Chapter  The chapter is mostly organized in a crescendo of difficulty of the proofs of each property. The proofs of duality and substitutivity are by induction; the proof of preservation hints at some of the structure of a realizability proof; and we end with strong normalization which is a full realizability proof with type interpretation contexts (for the interpretation of type variables). We begin with the extension of reduction we will study in this chapter, parallel reduction.

3.1 Parallel Reduction

§3.3. Syntactic Congruent Closure of Reduction  Parallel reduction is no more than allowing inner cuts—those other than the outermost one—to reduce. By analogy with equality, we also call this type of reduction congruent. In the Dual Calculus, where terms, co-terms and cuts are defined by mutual induction, we must carry on with reduction also within terms and co-terms (cf. table 3.1).\footnote{There is a design space to be navigated in choosing how to do this exactly. For example, it is fairly common in the lambda-calculus to restrict reduction inside lambda abstractions. Similarly, we explained negations of a co-term $k$ intuitively to be the encapsulation of the suspended execution of $k$ so that it could be moved around by a program just like a normal value and executed at a later date; in this reading, it would be natural not to perform reduction inside negations. This, however, would go against our goal of being as liberal as possible in our analysis of reduction.} This type of reduction is denoted by two
Table 3.1: Parallel reduction for the Dual Calculus

parallel reduction arrows \( \Rightarrow \). Note how in this reduction strategy a cut of an abstraction may cut in four different ways: reducing the abstraction; reducing the co-abstraction; reducing within the abstraction; and reducing inside the co-abstraction. Whilst the base of the inductive definition of parallel reduction is the non-deterministic reduction strategy, it could be easily adapted to either prioritizing strategy by changing that one case in the definition.

### 3.2 Duality

**§3.4. Duality for the Extension** Much has been made thus far about the importance of duality in the Dual Calculus—as if to stress this, duality is right there in the name. The duality for terms and co-terms, and the self-duality for cuts was already defined in table 2.10; table 3.2 expands it for the types in our second-order extension. (Note that for applications and their duals, the order of the arguments is swapped.) Recall that we assume that the duality for (co-) variables is given by some bijection that we also represent by \( \circ \) (as per §2.38).

**§3.5. Duality as a Model for Reduction** Our goal is to prove that reduction of the dual of any redex proceeds to the dual of the corresponding reduct. Stated differently, we want to show that the dualization provides a model of the original calculus. For the most part, the (head) reduction rules result in sub-phrases of the original phrases; the only difficulty lies in the reduction of abstractions which require substitution: we need to prove that duality is preserved by this operation. Because the duality interchanges terms with co-terms, we need to turn substitution for variables into substitution for co-variables—and vice-versa.
DUALITY

Types:
\[(A \rightarrow B)^{\circ} = B^{\circ} - A^{\circ}\]
\[(\forall X . T)^{\circ} = \exists X . T^{\circ}\]
\[(A - B)^{\circ} = B^{\circ} \rightarrow A^{\circ}\]
\[(\exists X . T)^{\circ} = \forall X . T^{\circ}\]

Phrases:
\[(\lambda x . t)^{\circ} = \mu x^{\circ} . (t^{\circ})\]
\[(\mu a . (k))^{\circ} = \lambda a^{\circ} . (k^{\circ})\]
\[(t @ k)^{\circ} = (k^{\circ} \# t^{\circ})\]
\[((t @ k))^{\circ} = (k^{\circ} @ t^{\circ})\]
\[(a(t))^{\circ} = e[t^{\circ}]\]
\[(a[k])^{\circ} = e(k^{\circ})\]
\[(e(t))^{\circ} = a[t^{\circ}]\]
\[(e[k])^{\circ} = a(k^{\circ})\]

Table 3.2: Duality for the arrow and quantifier types of DC.

Lemma 1 (Substitution). Substitution by terms and substitution by co-terms preserve the duality \((-)^{\circ}\):
\[(t[u/x])^{\circ} = t^{\circ}[u^{\circ}/x^{\circ}]\]
\[(k[u/x])^{\circ} = k^{\circ}[u^{\circ}/x^{\circ}]\]
\[(c[u/x])^{\circ} = c^{\circ}[u^{\circ}/x^{\circ}]\]

Proof. For each type of substitution, the proof is by mutual induction on the structure of the phrases. A few cases:

Variables:
\[(x[u/x])^{\circ} = u^{\circ} = x^{\circ}[u^{\circ}/x^{\circ}]\]

If \(t = y \neq x\), as the duality operation is assumed to be a bijection on variables, we have that \(x^{\circ} \neq y^{\circ}\); hence:
\[(y[u/x])^{\circ} = y^{\circ} = y^{\circ}[u^{\circ}/x^{\circ}]\]

For co-variables the situation is akin to that of differing variables:
\[(\beta[u/x])^{\circ} = \beta^{\circ} = \beta^{\circ}[u^{\circ}/x^{\circ}]\]
REDUCTION PROPERTIES OF THE DUAL CALCULUS

Cut: By the (mutual) induction hypothesis, for a cut $t \cdot k$:

\[ ((t \cdot k)[u/x])^o = (t[u/x] \cdot k[u/x])^o = (k[u/x])^o \cdot (t[u/x])^o = k^o[u^o/x^o] \cdot r^o[u^o/x^o] = (k^o \cdot r^o)[u^o/x^o] = (t \cdot k)^o[u^o/x^o] \]

Abstractions: For $k = y.(c)$ in the conditions of the lemma, we assume (up to $\alpha$-conversion) that the bound variable $y$ is not the $x$ under substitution nor in the free variables of $u$; therefore, $y^o \notin \text{fv}(u^o)$ since dualization preserves the free variables of a phrase (up to duals). We assume the same holds for bound co-variables $\beta$ in terms $t = \beta.(c)$. Thence:

\[
(y.(c)[u/x])^o = (y.(c[u/x]))^o = (\beta.(c[u/x]))^o = (\beta.(c)[u/x])^o
\]

\[
\]

Implication:

\[
(\lambda y.(t)[u/x])^o = (\lambda y.(t[u/x]))^o = ((t \cdot k)[u/x])^o = ((t[u/x] \cdot k[u/x])^o
\]

\[
\]

Subtraction:

\[
((t \# k)[u/x])^o = ((t[u/x] \# k[u/x])^o = (\mu \beta.(t)[u/x])^o = (\mu \beta.(t[u/x])^o
\]

\[
\]

Theorem 4. The duality $(-)^o$ preserves head reduction:

\[ c \leftrightarrow d \implies c^o \leftrightarrow d^o. \]

50
DUALITY

Proof. By simple case analysis. For abstractions, we make use of the substitution lemma for duality (lemma 1).

\[(\lambda x . (t \cdot (u @ k)))^\circ = (u @ k)^\circ \cdot (\lambda x . (t))^\circ = (k^\circ \cdot u^\circ) \cdot u^\circ \cdot (t^\circ) \Rightarrow k^\circ \cdot t^\circ [u^\circ / x^\circ] = k^\circ \cdot (t[u/x])^\circ = (t[u/x] \cdot k)^\circ\]

\[((u\#l) \cdot \mu \alpha . (k))^\circ = (\mu \alpha . (k))^\circ \cdot (u\#l)^\circ = \lambda \alpha^\circ . (k)^\circ \cdot (l^\circ @ u^\circ) \Rightarrow k^\circ \cdot l^\circ [\alpha^\circ / u^\circ] = (k[l/\alpha])^\circ \cdot u^\circ = (u \cdot k[l/\alpha])^\circ\]

\[(t \cdot x . (c))^\circ = (x . (c))^\circ \cdot t^\circ = x^\circ . (c^\circ) \cdot t^\circ \Rightarrow c^\circ [t^\circ / x^\circ] = (c[t/x])^\circ \]

\[(\alpha . (c) \cdot k)^\circ = k^\circ \cdot (\lambda x . (t))^\circ = k^\circ \cdot (\alpha . (c))^\circ \Rightarrow c^\circ [k^\circ / \alpha^\circ] = (c[k/\alpha])^\circ\]

\[\square\]

Theorem 5. The duality \((-)^\circ\) preserves parallel reduction: for terms \(t\) and \(u\), co-terms \(k\) and \(l\), and cuts \(c\) and \(d\),

\[t \Rightarrow u \Rightarrow t^\circ = u^\circ \quad k \Rightarrow l \Rightarrow k^\circ = l^\circ\]

\[c \Rightarrow d \Rightarrow c^\circ = d^\circ\]

Proof. Proof by mutual induction on the definition of parallel reduction.

Head reductions: Theorem 4 implies immediately the base case.

Abstractions: Assume \(c \Rightarrow d\); by induction hypothesis \(c^\circ \Rightarrow d^\circ\), and therefore, respectively:

\[(\alpha . (c))^\circ = \alpha^\circ . (c^\circ) \Rightarrow \alpha^\circ . (d^\circ) = (\alpha . (d))^\circ\]

\[(x . (c))^\circ = x^\circ . (c^\circ) \Rightarrow x^\circ . (d^\circ) = (x . (d))^\circ\]

51
Cuts:

\[(t \cdot m)^\circ = m^\circ \cdot t^\circ \]
\[\iff m^\circ \cdot u^\circ \]
\[= (u \cdot m)^\circ \]

\[(v \cdot k)^\circ = k^\circ \cdot v^\circ \]
\[\iff l^\circ \cdot v^\circ \]
\[= (v \cdot l)^\circ \]

The remaining cases follow similarly.  

3.3 Substitutivity

§3.6. Indistinguishability  If two things are equal then they should not be distinguishable by any act one might do with them or on them: they should be equal no matter what context they are put in; and should remain equal whenever they are manipulated in a uniform manner. In formal languages, and generalizing equality to binary relations, this property is called substitutivity (see, e.g., Schwichtenberg and Wainer, 2011, section 1.2). The name derives from the fact that—syntactically—putting a phrase in a context is to substitute it in a phrase that represents that context; and to instantiate it with some other phrase is to substitute one of the former’s (free) variables by the latter. Here, we deal with head and parallel reduction—these are the relations we are interested in. The intuition that we are trying to capture is that redexes and reducts behave, essentially, the same.

Theorem 6 (Head Substitutivity I). For cuts \(c\) and \(d\), any terms \(v\) and any co-terms \(m\), and any co-variable \(\alpha\) and variable \(x\),

\[c \step d \implies c \maysub{v}{x} \step d \maysub{v}{x}\]
\[c \step d \implies c \maysub{m/\alpha} \step d \maysub{m/\alpha}\]

Proof. Each case, whether we substitute a variable or a co-variable, is proved by case analysis on the definition of head reduction. The only interesting cases are those dealing with abstractions where reduction introduces yet another substitution.

For the structural abstractions we need to assume, as usual, that \(v\) does not have free any of the bound variables in in the cut, and that \(x\) does not match any of the bound variables in the cut under substitution. In these conditions we get, in particular, that for any term \(u\)

\[c \maysub{u/y} \maysub{v/x} = c \maysub{v/x} \maysub{u/v/x} / y\]

and, thence,

\[\beta.(c \cdot k)\maysub{v/x} = \beta.(c \maysub{v/x}) \cdot k \maysub{v/x}\]
\[= c \maysub{v/x} \cdot k \maysub{v/x} / \beta\]
\[= c \maysub{u/y} \maysub{v/x}\]
§3.7. Substitutivity and Commutativity

It is easier to motivate the relevance of substitutivity for parallel reduction: head reduction happens only in one place; parallel reduction can happen everywhere, concurrently. It is important (as we shall see) that the choice of order in which one applies the congruent rules is immaterial to the outcome of reduction. Assume, $c \equiv d$; then a cut with an abstraction on $c$ can reduce by either the head rule or the parallel rule:

$$t \cdot x.(c) \Rightarrow c[t/x]$$
$$\Downarrow\quad \Downarrow\quad \Downarrow$$
$$t \cdot x.(d) \Rightarrow d[t/x]$$

Substitutivity implies that both choices are equivalent, in the sense that they can both be made to converge. Of course, at the parallel level, reduction happens not only at the level of cuts, but also of terms and, so, the statement of this form of substitutivity must be accordingly adapted.

**Theorem 7 (Parallel Substitutivity I).** For terms $t$ and $t_i$, co-terms $k$, $k_i$, cuts $c_i (i \in \{0, 1\})$, and (co-) variables $x$ and $a$, we have that

$$t_0 \equiv t_1 \implies t_0[t/x] \equiv t_1[t/x] \quad t_0 \equiv t_1 \implies t_0[k/a] \equiv t_1[k/a]$$
$$k_0 \equiv k_1 \implies k_0[t/x] \equiv k_1[t/x] \quad k_0 \equiv k_1 \implies k_0[k/a] \equiv k_1[k/a]$$
$$c_0 \equiv c_1 \implies c_0[t/x] \equiv c_1[t/x] \quad c_0 \equiv c_1 \implies c_0[k/a] \equiv c_1[k/a]$$

**Proof.** The proof is by induction on the definition of parallel reduction.

The base case, substitutivity for head reduction, is essentially theorem 4:

$$c \equiv d \implies c[t/x] \equiv d[t/x]$$

For the inductive step on the definition of congruent reduction, there are many, many, cases to consider. The only ones that are slightly more difficult are those involving abstractions which require, as usual—taking phrases up to $\alpha$-equivalence so that there are no clashes between bound-variables, and the variable being substituted and those free variables in the term being substituted in.

$$\alpha.(c_0)[t/x] \quad \quad y.(c_0)[t/x]$$
$$= \alpha.(c_0[t/x]) \quad \quad = y.(c_0[t/x])$$
$$\equiv \alpha.(c_i[t/x]) \quad \equiv y.(c_i[t/x])$$
$$= \alpha.(c_i)[t/x] \quad \quad \equiv y.(c_i)[t/x]$$

The congruent cases for cuts are representative of the remaining ones.

$$(t_0 \cdot k')[t/x] \quad \quad (t' \cdot k_0)[t/x]$$
$$= t_0[t/x] \cdot k'[t/x] \quad \quad = t'[t/x] \cdot k_0[t/x]$$
$$\equiv t_1[t/x] \cdot k'[t/x] \quad \equiv t'[t/x] \cdot k_1[t/x]$$
$$= (t_1 \cdot k')[t/x] \quad \quad = (t' \cdot k_1)[t/x]$$

\[\square\]

---

\[\square\] The situation described here should not be confused with the failure of confluence for the non-deterministic head reduction rules (as per [§2.4]). There we have two conflicting rules living at the same level; here, the rules that we show commute live one at the base level, and the other at the level of the congruent closure.
§3.8. Contextualization The other form of substitutivity is the one where reduction happens on the phrase being substituted in. The phrase in which we substitute acts as a context of sorts and the question is if the reduction of that phrase still applies within. A few nuances apply: As the substitution only happens for terms and co-terms this form is trivially satisfied by head reduction for, there, reduction happens only on cuts. As is the case in intuitionistic calculi (e.g. [Schwichtenberg and Wainer, 2011, 1.2.1]), we need to consider the transitive and reflexive closure of the reduction relation \( \Rightarrow \); the number of reduction steps actually needed reflects the number of appearances of the variable being substituted—each appearance requires one reduction. First, though, we need a lemma.

Lemma 2. The syntactic operators of the Dual Calculus preserve the reflexive transitive closure of parallel reduction:

\[
\begin{align*}
\langle t_0, t \rangle \Rightarrow^* & \langle t_1, t \rangle & \langle t, t_0 \rangle \Rightarrow^* & \langle t, t_1 \rangle & t_1(t_0) \Rightarrow^* & t_1(t_1) & t_2(t_0) \Rightarrow^* & t_2(t_1) & \lambda x. (t_0) \Rightarrow^* & \lambda x. (t_1) & (t_0 \# k) \Rightarrow^* & (t_1 \# k) & a(t_0) \Rightarrow^* & a(t_1) & e(t_0) \Rightarrow^* & e(t_1) & \text{not } [t_0] \Rightarrow^* & \text{not } [t_1] & t_0 \cdot k \Rightarrow^* & t_1 \cdot k & k_0 \Rightarrow^* & k_1 \\
\end{align*}
\]

\[
\begin{align*}
\langle t_0, t \rangle = & \langle t_1, t \rangle & \langle t, t_0 \rangle = & \langle t, t_1 \rangle & t_1(t_0) = & t_1(t_1) & t_2(t_0) = & t_2(t_1) & \lambda x. (t_0) = & \lambda x. (t_1) & (t_0 \# k) = & (t_1 \# k) & a(t_0) = & a(t_1) & e(t_0) = & e(t_1) & \text{not } [t_0] = & \text{not } [t_1] & t_0 \cdot k = & t_1 \cdot k & k_0 = & k_1 \\
\end{align*}
\]

\[
\begin{align*}
c_0 \Rightarrow^* c_1 & \Rightarrow x. (c_0) \Rightarrow^* x. (c_1) & c_0 = & c_1 & x. (c_0) = & x. (c_1) \\
\end{align*}
\]

Proof. We prove this by induction on the number of reductions of the phrase in the antecedent of each statement. The base case is, by reflexivity, equality:

\[
\begin{align*}
\langle t_0, t \rangle = & \langle t_1, t \rangle & \langle t, t_0 \rangle = & \langle t, t_1 \rangle & t_1(t_0) = & t_1(t_1) & t_2(t_0) = & t_2(t_1) & \lambda x. (t_0) = & \lambda x. (t_1) & (t_0 \# k) = & (t_1 \# k) & a(t_0) = & a(t_1) & e(t_0) = & e(t_1) & \text{not } [t_0] = & \text{not } [t_1] & t_0 \cdot k = & t_1 \cdot k & k_0 = & k_1 \\
\end{align*}
\]

The inductive step, corresponding to transitivity, is dealt with by separating a reduction of length \( n + 1 \) into one of length \( n \) (to which the induction hypothesis applies) followed
by one ordinary step.

\[
\begin{align*}
t_0 \Rightarrow^n u \Rightarrow t_1 \Rightarrow \\
\{ \langle t_0, t \rangle \Rightarrow^n \langle u, t \rangle \Rightarrow \langle t_1, t \rangle \\
\langle t, t_0 \rangle \Rightarrow^n \langle t, u \rangle \Rightarrow \langle t, t_1 \rangle \\
t_1(t_0) \Rightarrow^n t_1(u) \Rightarrow t_1(t_1) \\
t_2(t_0) \Rightarrow^n t_2(u) \Rightarrow t_2(t_1) \\
\lambda x.(t_0) \Rightarrow^n \lambda x.(u) \Rightarrow \lambda x.(t_1) \\
(t_0 \# k) \Rightarrow^n (u \# k) \Rightarrow (t_1 \# k) \\
a(t_0) \Rightarrow^n a(u) \Rightarrow a(t_1) \\
e(t_0) \Rightarrow^n e(u) \Rightarrow e(t_1) \\
\text{not}(t_0) \Rightarrow^n \text{not}(u) \Rightarrow \text{not}(t_1) \\
t_0 \cdot k \Rightarrow^n u \cdot k \Rightarrow t_1 \cdot k
\end{align*}
\]

\[
\begin{align*}
k_0 \Rightarrow^n l \Rightarrow k_1 \Rightarrow \\
\{ \text{fst}[k_0] \Rightarrow^n \text{fst}[l] \Rightarrow \text{fst}[k_1] \\
\text{snd}[k_0] \Rightarrow^n \text{snd}[l] \Rightarrow \text{snd}[k_1] \\
[k, k_0] \Rightarrow^n [k, l] \Rightarrow [k, k_1] \\
[k_0, k] \Rightarrow^n [l, k] \Rightarrow [k_1, k] \\
\mu \alpha.(k_0) \Rightarrow^n \mu \alpha.(l) \Rightarrow \mu \alpha.(k_1) \\
t \# k_0 \Rightarrow^n (t \# l) \Rightarrow (t \# k_1) \\
a[k_0] \Rightarrow^n a[l] \Rightarrow a[k_1] \\
e[k_0] \Rightarrow^n e[l] \Rightarrow e[k_1] \\
\text{not}(k_0) \Rightarrow^n \text{not}(l) \Rightarrow \text{not}(k_1) \\
t \cdot k_0 \Rightarrow^n t \cdot l \Rightarrow t \cdot k_1
\end{align*}
\]

\[
\begin{align*}
c_0 \Rightarrow^n d \Rightarrow c_1 \Rightarrow \alpha.(c_0) \Rightarrow^n \alpha.(d) \Rightarrow \alpha.(c_1) \\
c_0 \Rightarrow^n d \Rightarrow c_1 \Rightarrow x.(c_0) \Rightarrow^n x.(d) \Rightarrow x.(c_1)
\end{align*}
\]

\[\Box\]

**Theorem 8 (Substitutivity II).** Substituting in preserves parallel reduction:

\[
\begin{align*}
t_0 \Rightarrow t_1 \Rightarrow \\
\{ t_{[t_0/x]} \Rightarrow t_{[t_1/x]} \\
k_{[t_0/x]} \Rightarrow k_{[t_1/x]} \\
c_{[t_0/x]} \Rightarrow c_{[t_1/x]}
\end{align*}
\]

\[
\begin{align*}
k_0 \Rightarrow k_1 \Rightarrow \\
\{ t_{[k_0/a]} \Rightarrow t_{[k_1/a]} \\
k_{[k_0/a]} \Rightarrow k_{[k_1/a]} \\
c_{[k_0/a]} \Rightarrow c_{[k_1/a]}
\end{align*}
\]

**Proof.** We prove the claims by mutual induction on the structure of DC phrases, and for each kind of substitution.

For variables, we distinguish two cases, depending on whether the variable in question is the one being substituted or not. If it is, we get

\[x_{[t_0/x]} = t_0 \Rightarrow t_1 = x_{[t_1/x]};\]

for any other variable \(y\), we get

\[y_{[t_0/x]} = y = y_{[t_1/x]};\]

55
REDUCTION PROPERTIES OF THE DUAL CALCULUS

this, because we are considering the reflexive closure of congruent reduction, validates the statement of the theorem. Co-variables follow similarly,

\[ \beta[t_0/x] = \beta = \beta[t_1/x]. \]

The conjunctive (term) case neatly shows the need for pairing the induction hypothesis with the transitive closure.

\[
\langle t, t'\rangle[t_0/x] = \langle t[t_0/x], t'[t_0/x] \rangle \\
\Rightarrow^* \langle t[t_1/x], t'[t_0/x] \rangle \\
\Rightarrow^* \langle t[t_1/x], t'[t_1/x] \rangle \\
= \langle t, t'\rangle[t_1/x]
\]

\[
\text{fst}[k][t_0/x] = \text{fst}[k][t_0/x] \\
\Rightarrow^* \text{snd}[k][t_0/x] = \text{snd}[k][t_0/x] \\
\Rightarrow^* \text{snd}[k][t_1/x] = \text{snd}[k][t_1/x]
\]

Abstractions require the use of \(\alpha\)-equivalence to prevent illicit binding of (co-) variables; for variety we show the implicational and subtractive cases:

\[
\lambda y.(k)[t_0/x] = \lambda y.(k)[t_0/x] \\
\Rightarrow^* \lambda y.(k)[t_1/x] = \lambda y.(k)[t_1/x] \\
\Rightarrow^* (t @ k)[t_0/x] = (t @ k)[t_0/x] \\
= (t @ k)[t_0/x] \\
= (t @ k)[t_0/x] \\
= (t @ k)[t_1/x] \\
= (t @ k)[t_1/x] \\
\]

The remaining cases are similar. \(\Box\)

§3.9. And Head Reduction? To conclude this section we remark that head reduction would trivially satisfy a result like theorem 8 there, there is no reduction for either terms and co-terms and the preconditions of the theorem are trivially satisfied.
3.4 Preservation

§3.10. General Statement  From a programming point of view, types exist to guarantee safety properties in programming languages—we recall the well-known mantra that well-typed programs don’t go wrong. In particular, if a program is well-typed, its execution maintains the invariant of not going wrong—and, so, the reduct should also be well-typed. This property is usually referred to as Preservation or Subject Reduction. The object of this section is to prove exactly this—that even in a Classical world, well-typed programs remain so during reduction and, so, “don’t go wrong.”

§3.11. A Different Cut  The typing rule for DC cuts (table 2.6) was defined as having different typing (co-) contexts for each assumption sequent. This was inspired directly by the cut rule of LK since the previous chapter dealt with the comparison between the two systems. Having a rule that so closely mimicked the original simplified this comparison. In the sequel we shall use the following variant:

\[ \frac{\Gamma \vdash t : A \mid \Delta \quad \Gamma \mid k : A \vdash \Delta}{\Gamma \vdash t \cdot k \vdash \Delta} \]

It is not hard to see that this rule types the same cuts as the original one—some judicious, if laborious, use of contraction being necessary. By using this rule, we can reduce the number of parameters we need to deal with in our proofs.

§3.12. Proof Outline  The obvious—and, as we shall see, correct—way of proving Preservation relies on case analysis on the head reduction relation. However, several lemmas shall be needed which are easier to motivate if one ponders how one might go around doing the proof. Let’s consider the reduction of a pair with a projection

\[ \Gamma \vdash \langle t, t' \rangle \cdot \text{fst}[k] \vdash \Delta. \]

Quite obviously each of the sub-phrases has a type of the form \( A \land B \); the type is determined by the form of the sub-phrases. We make this explicit in remark 4. This leads us to

\[ \Gamma \vdash \langle t, t' \rangle : A \land B \mid \Delta \quad \text{and} \quad \Gamma \mid \text{fst}[k] : A \land B \vdash \Delta; \]

from where, again appealing to the same remark, we can determine that

\[ \Gamma \vdash t : A \mid \Delta \quad \text{and} \quad \Gamma \mid k : A \vdash \Delta \]

and using the modified cut rule (§3.11) we conclude that

\[ \Gamma \vdash t \cdot k \vdash \Delta. \]

The abstraction rules and the right (resp. left) universal (resp. existential) quantification rule follow the same general pattern but require some extra lemmas to account for their structure. For abstraction we need preservation under substitution (lemma 4). For the quantification rules, in the process of eliminating the quantifier we must be able to replace the quantified type variable by any other type—hence, we have corollary 2 which follows from the more general lemma 3.
Lemma 3. For \( \vec{X} \) a vector \( X_1, \ldots, X_n \) of \( n \) distinct type variables, let \( \Gamma(\vec{X}) \) and \( \Delta(\vec{X}) \) stand for context schemas where free occurrences of the variables \( \vec{X} \) in typing contexts \( \Gamma \) and \( \Delta \) are abstracted. Then, any valid typing of terms \( t \), co-terms \( k \) or cuts \( c \) on arbitrary \( \Gamma(\vec{X}) \) and \( \Delta(\vec{X}) \) yields a valid typing whenever the type variables \( \vec{X} \) are replaced by types \( \vec{T} = T_1, \ldots, T_n \) that do not have free any of the bound variables in the typescheme \( F(\vec{X}) \)

\[
\begin{align*}
\Gamma(\vec{X}) & \vdash t : F(\vec{X}) \mid \Delta(\vec{X}) \quad \Rightarrow \quad \Gamma(\vec{T}) \vdash t : F(\vec{T}) \mid \Delta(\vec{T}) \\
\Gamma(\vec{X}) & \vdash k : F(\vec{X}) \rightarrow \Delta(\vec{X}) \quad \Rightarrow \quad \Gamma(\vec{T}) \vdash k : F(\vec{T}) \rightarrow \Delta(\vec{T}) \\
\Gamma(\vec{X}) & \vdash c \not\vdash \Delta(\vec{X}) \quad \Rightarrow \quad \Gamma(\vec{T}) \vdash c \not\vdash \Delta(\vec{T})
\end{align*}
\]

Proof. We reason by mutual induction on the structure of the typing derivations.

Base cases:

\[
\begin{align*}
x : F(\vec{T}) & \vdash x : F(\vec{T}) \\
| \alpha : F(\vec{T}) & \vdash \alpha : F(\vec{T})
\end{align*}
\]

Operational rules: Via the induction hypothesis, we can (for the most part) easily derive the lemma for the operational rules.

\[
\begin{align*}
\frac{(\Gamma(\vec{X}) \vdash t : F(\vec{X}) \mid \Delta(\vec{X}))}{\Gamma(\vec{T}) \vdash t : F(\vec{T}) \mid \Delta(\vec{T})} \quad & \text{IH} \\
\frac{(\Gamma(\vec{X}) \vdash t : G(\vec{X}) \mid \Delta(\vec{X}))}{\Gamma(\vec{T}) \vdash t : G(\vec{T}) \mid \Delta(\vec{T})} \quad & \text{IH} \\
\frac{(\Gamma(\vec{T}) \vdash t_1(t) : F(\vec{T}) \lor G(\vec{T}) \mid \Delta(\vec{T}))}{\Gamma(\vec{T}) \vdash t_1(t) : F(\vec{T}) \lor G(\vec{T}) \mid \Delta(\vec{T})} \\
\frac{(\Gamma(\vec{T}) \vdash t_2(t) : (F \lor G)(\vec{T}) \mid \Delta(\vec{T}))}{\Gamma(\vec{T}) \vdash t_2(t) : (F \lor G)(\vec{T}) \mid \Delta(\vec{T})} \\
\frac{(\Gamma(\vec{X}) \vdash k : F(\vec{X}) \not\vdash \Delta(\vec{X}))}{\Gamma(\vec{T}) \vdash k : F(\vec{T}) \not\vdash \Delta(\vec{T})} \quad & \text{IH} \\
\frac{(\Gamma(\vec{X}) \vdash k' : G(\vec{X}) \not\vdash \Delta(\vec{X}))}{\Gamma(\vec{T}) \vdash k' : G(\vec{T}) \not\vdash \Delta(\vec{T})} \quad & \text{IH} \\
\frac{(\Gamma(\vec{T}) \vdash [k,k'] : F(\vec{T}) \lor G(\vec{T}) \not\vdash \Delta(\vec{T}))}{\Gamma(\vec{T}) \vdash [k,k'] : (F \lor G)(\vec{T}) \not\vdash \Delta(\vec{T})}
\end{align*}
\]

As we unpack a type abstraction, a new free type variable appears in the schema—we, thus, extend our typeschemes with an extra type variable \( Y \) to \( F(\vec{X}, Y) \). To respect the side-condition of the right rule of universal quantification, we use the pre-condition on \( T \), so that we are guaranteed that the abstracted type variable \( Y \) does not appear in \( \Gamma(\vec{T}) \) nor in \( \Delta(\vec{T}) \):

\[
\begin{align*}
\frac{(\Gamma(\vec{X}) \vdash t : F(\vec{X}, Y) \mid \Delta(\vec{X}))}{\Gamma(\vec{T}) \vdash t : F(\vec{T}, Y) \mid \Delta(\vec{T})} \quad & \text{IH} \\
\frac{(\Gamma(\vec{X}) \vdash k : F(\vec{X}, T') \not\vdash \Delta(\vec{X}))}{\Gamma(\vec{T}) \vdash k : F(\vec{T}, T') \not\vdash \Delta(\vec{T})} \quad & \text{IH} \\
\frac{(\Gamma(\vec{T}) \vdash t : \forall Y.F(\vec{T}, Y) \mid \Delta(\vec{T}))}{\Gamma(\vec{T}) \vdash t : \forall Y.F(\vec{T}, Y) \mid \Delta(\vec{T})} \\
\frac{(\Gamma(\vec{T}) \vdash k : F(\vec{T}, T') \not\vdash \Delta(\vec{T}))}{\Gamma(\vec{T}) \vdash k : F(\vec{T}, T') \not\vdash \Delta(\vec{T})}
\end{align*}
\]

Similar derivations can be used to establish the induction for the operational rules. The cut and structural rules offer no particular difficulties. \( \square \)
Corollary 2. Let \( X \) be a type variable that does not appear free in \( \Gamma \), nor in \( \Delta \). Then, any valid typing that depends on \( X \) yields a valid typing when \( X \) is replaced by any other type \( T \):

\[
\Gamma \vdash t : F(X) | \Delta \implies \Gamma \vdash t : F(T) | \Delta
\]

\[
\Gamma \vdash k : F(X) \vdash \Delta \implies \Gamma \vdash k : F(T) \vdash \Delta
\]

Proof. Follows from lemma 3 by noting that, since \( X \) does not appear free in \( \Gamma \) (resp. \( \Delta \)), \( \Gamma(T) = \Gamma \) (resp. \( \Delta(T) = \Delta \)). \(\square\)

Remark 1. The syntactic operators determine the structure of their admissible types:

\[
\Gamma \vdash \langle t, u \rangle : T | \Delta \implies T = A \land B, \Gamma \vdash t : A | \Delta \text{ and } \Gamma \vdash u : B | \Delta
\]

\[
\Gamma \mid \text{fst}[k] : T \vdash \Delta \implies T = A \land B \text{ and } \Gamma \mid k : A \vdash \Delta
\]

\[
\Gamma \vdash \lambda x.(t) : T | \Delta \implies T = A \rightarrow B \text{ and } x : A, \Gamma \vdash t : B | \Delta
\]

\[
\Gamma \vdash (t @ k) : T | \Delta \implies T = A \rightarrow B, \Gamma \vdash t : A | \Delta \text{ and } \Gamma \mid k : B \vdash \Delta
\]

\[
\Gamma \vdash (t \# k) : T | \Delta \implies T = A \rightarrow B, \Gamma \vdash t : A | \Delta \text{ and } \Gamma \mid k : B \vdash \Delta
\]

\[
\Gamma \vdash \mu \alpha.(k) : T | \Delta \implies T = A \rightarrow B \text{ and } \Gamma \vdash t : A | \Delta, \alpha : B
\]

\[
\Gamma \vdash a(t) : T | \Delta \implies T = \forall X . F(X) \text{ and } \Gamma \vdash t : F(X) | \Delta
\]

\[
\Gamma \mid a[k] : T \vdash \Delta \implies T = \forall Y . F(Y) \text{ and } \Gamma \mid k : F(A) \vdash \Delta
\]

\[
\Gamma \vdash e(t) : T | \Delta \implies T = \exists Y . F(Y) \text{ and } \Gamma \vdash t : F(A) | \Delta
\]

\[
\Gamma \mid e[k] : T \vdash \Delta \implies T = \exists X . F(X) \text{ and } \Gamma \mid k : F(X) \vdash \Delta
\]

\[
\Gamma \vdash a.(c) : T | \Delta \implies \Gamma \vdash c \vdash \Delta, \alpha : T
\]

\[
\Gamma \mid x.(c) : T \vdash \Delta \implies x : T, \Gamma \vdash c \vdash \Delta
\]

\[
\Gamma \vdash t \bullet k \vdash \Delta \implies \Gamma \vdash t : A | \Delta \text{ and } \Gamma \mid k : A \vdash \Delta
\]

for some \( A, B \) and \( F(X) \) with \( X \) not free in the relevant typing (co-) contexts.

§3.13. Simultaneous Substitution  By a simultaneous substitution \( \sigma : (\Gamma \vdash \Delta) \mapsto (\Gamma' \vdash \Delta') \) we understand an assignment of variables and co-variables in \( \Gamma \) and \( \Delta \) to terms and co-terms (resp.) of the same type under contexts \( \Gamma' \) and \( \Delta' \). Formally:

\[
\Gamma = (\Gamma_0, x : A, \Gamma_y) \implies \Gamma' \vdash \sigma(x) : A | \Delta'
\]

\[
\Delta = (\Delta_0, \alpha : A, \Delta_y) \implies \Gamma' \mid \sigma(\alpha) : A \vdash \Delta'.
\]

Application of a substitution is done in the obvious manner and is denoted by \(-[\sigma]\).

Two immediate examples of simultaneous substitution are the identity, and ordinary substitution.
Lemma 4. Substitutions on well-typed phrases induce well-typed phrases: if we have some
\[ \Gamma \vdash t : A | \Delta, \quad \Gamma | k : A \vdash \Delta, \quad \text{or} \quad \Gamma \vdash c \vdash \Delta \]
then for any \( \Gamma' \) and \( \Delta' \) where, up to \( \alpha \)-conversion of types, no bound type variables in \( A \) appear free, and
\[ \sigma : (\Gamma \vdash \Delta) \mapsto (\Gamma' \vdash \Delta') \]
we have, respectively, that
\[ \Gamma' \vdash t[\sigma] : A | \Delta', \quad \Gamma' | k[\sigma] : A \vdash \Delta', \quad \text{or} \quad \Gamma' \vdash c[\sigma] \vdash \Delta'. \]

Proof. The proof is by mutual induction on the typing rules for the phrase where the substitution occurs. The quantification on the substitutions is strictly necessary for the contexts and (necessarily) the substitutions used in the proof are not constant: contraction and the abstractions require applying the induction hypothesis to an extended substitution for extended contexts.

The base case is trivial by the definition of \( \sigma : (\Gamma \vdash \Delta) \mapsto (\Gamma' \vdash \Delta') \).

For the operational rules we have, e.g.:

\[
\begin{align*}
\Gamma' & \vdash t[\sigma] : A | \Delta' \\
\Gamma' & \vdash t[\sigma] : A | \Delta' & (\Gamma \vdash t : B | \Delta) \\
\Gamma' & \vdash t[\sigma] : B | \Delta' & (\Gamma \vdash t : B | \Delta) \\
\Gamma' & \vdash t[\sigma] : B | \Delta' & (\Gamma \vdash t : B | \Delta) \\
\Gamma' & \vdash t[\sigma] : B | \Delta' & (\Gamma \vdash t : B | \Delta) \\
\end{align*}
\]

For the universal quantification term rule we need to take care, additionally, of the side condition. Luckily, the additional condition on contexts \( \Gamma' \) and \( \Delta' \) comes to our rescue, guaranteeing that the bound variable does not appear free in the typing contexts of the new term. (The same applies to the left existential quantification rule.)

\[
\begin{align*}
\Gamma' & \vdash t[\sigma] : F(X) | \Delta' \\
\Gamma' & \vdash t[\sigma] : F(X) | \Delta' & (\Gamma \vdash t : F(T) \vdash \Delta) \\
\Gamma' & \vdash t[\sigma] : F(T) | \Delta' & (\Gamma \vdash t : F(T) \vdash \Delta) \\
\Gamma' & \vdash t[\sigma] : F(T) | \Delta' & (\Gamma \vdash t : F(T) \vdash \Delta) \\
\Gamma' & \vdash t[\sigma] : F(T) | \Delta' & (\Gamma \vdash t : F(T) \vdash \Delta) \\
\Gamma' & \vdash t[\sigma] : F(T) | \Delta' & (\Gamma \vdash t : F(T) \vdash \Delta) \\
\end{align*}
\]
For abstractions, assuming, as usual, no binding of free variables, we have that

\[ y. (c)[\sigma] = y. (c[\sigma[y \mapsto y]]). \]

The right-hand side substitution satisfies the conditions for being a substitution of type \( y : A, (\Gamma \vdash \Delta) \mapsto (y : A, \Gamma' \vdash \Delta') \). Hence,

\[
\frac{\Gamma \vdash c \vdash \Delta, a : A}{\Gamma' \vdash c[\sigma[a \mapsto a]] \vdash \Delta', a : A} \quad \text{(IH)} \\
\frac{\Gamma' \vdash a. (c[\sigma[a \mapsto a]]) \mid \Delta'}{\Gamma' \vdash a. (c)[\sigma] \mid \Delta'}
\]

For thinning, we observe that, e.g. a substitution \( \sigma : (x : A, \Gamma) \vdash (\Gamma' \vdash \Delta') \) satisfies the conditions for \((\Gamma \vdash \Delta) \mapsto (\Gamma' \vdash \Delta')\); and, therefore,

\[
\frac{\Gamma \vdash t : B \vdash \Delta}{\Gamma' \vdash t[\sigma] : B \vdash \Delta'} \quad \text{(IH)} \\
\frac{\Gamma' \vdash t[\sigma] : B \mid \Delta'}{\Gamma' \vdash t[\sigma] : B \mid \Delta'} \quad \text{(IH)}
\]

For contraction—arguing for once on the co-term side—we are given a substitution

\[ \sigma : (\Gamma \vdash (\Delta, a : A)) \mapsto (\Gamma' \vdash \Delta'); \]

in particular \( \Gamma' \vdash \sigma(a) : A \mid \Delta \), so that we can extend \( \sigma \) to fit the pre-contraction situation:

\[ \sigma[\beta \mapsto \sigma(a)] : (\Gamma \vdash (\Delta, \beta : A, a : A)) \mapsto (\Gamma' \vdash \Delta). \]

We then reason

\[
\frac{(x : A, y : B, \Gamma \vdash c \vdash \Delta, \lambda) \quad (\Gamma \vdash c \vdash \Delta, \beta : A, a : A)}{\Gamma' \vdash c[\sigma[y \mapsto \sigma(x)] \vdash \Delta'] \quad \text{(IH)}} \\
\frac{\Gamma' \vdash c[\sigma[y \mapsto \sigma(x)] \vdash \Delta']}{\Gamma' \vdash c[\sigma[y \mapsto \sigma(x)]] \vdash \Delta'}
\]

Interchange is easy as any substitution

\[ \sigma : (\Gamma \vdash \Delta'', \alpha_{i+1} : A_{i+1}, \alpha_i : A_i, \Delta) \mapsto (\Gamma' \vdash \Delta') \]

is also a substitution

\[ \sigma : (\Gamma \vdash \Delta'', \alpha_i : A_i, \alpha_{i+1} : A_{i+1}, \Delta) \mapsto (\Gamma' \vdash \Delta'). \]

\[ \square \]

**Theorem 9** (Head Reduction Preservation). *Reduction in DC preserves types:*

if \( \Gamma \vdash c \vdash \Delta \) and \( c \leadsto c' \) then \( \Gamma \vdash c' \vdash \Delta \).
REDUCTION PROPERTIES OF THE DUAL CALCULUS

Proof. The proof here is by case analysis on the reduction rules. For the operational rules we make heavy use of remark 1.

\[
\Gamma \vdash \langle t, t' \rangle \cdot fst[k] \vdash \Delta \\
\implies \Gamma \vdash \langle t, t' \rangle : A \wedge B \mid \Delta \text{ and } \Gamma \mid fst[k] : A \wedge B \vdash \Delta \\
\implies \Gamma \vdash t : A \mid \Delta \text{ and } \Gamma \mid k : A \vdash \Delta \\
\implies \Gamma \vdash t \cdot k \vdash \Delta
\]

\[
\Gamma \vdash \langle t, t' \rangle \cdot snd[k] \vdash \Delta \\
\implies \Gamma \vdash \langle t, t' \rangle : A \wedge B \mid \Delta \text{ and } \Gamma \mid snd[k] : A \wedge B \vdash \Delta \\
\implies \Gamma \vdash t' : B \mid \Delta \text{ and } \Gamma \mid k : B \vdash \Delta \\
\implies \Gamma \vdash t' \cdot k \vdash \Delta
\]

For the implication, subtraction, and abstraction rule we use, additionally, lemma 4. Take \(\sigma : (\Gamma \vdash \Delta) \mapsto (\Gamma \vdash \Delta)\) to denote the identity substitution for contexts \(\Gamma\) and \(\Delta\), and extend it with \(k\) for \(\alpha\):

\[
\Gamma \vdash \alpha.(c) \cdot k \vdash \Delta \\
\implies \Gamma \vdash \alpha.(c) : A \mid \Delta \text{ and } \Gamma \mid k : A \vdash \Delta \\
\implies \Gamma \vdash c \vdash \Delta, \alpha : A \text{ and } \Gamma \mid k : A \vdash \Delta \\
\implies \Gamma \vdash c[\sigma[\alpha \mapsto k]] \vdash \Delta \\
\implies \Gamma \vdash c[k/\alpha] \vdash \Delta
\]

\[
\Gamma \vdash t \cdot x.(c) \vdash \Delta \\
\implies \Gamma \vdash x.(c) : A \mid \Delta \text{ and } \Gamma \vdash t : A \mid \Delta \\
\implies x : A, \Gamma \vdash c \vdash \Delta \text{ and } \Gamma \vdash t : A \mid \Delta \\
\implies \Gamma \vdash c[\sigma[t/x]] \vdash \Delta \\
\implies \Gamma \vdash c[t/x] \vdash \Delta
\]

Theorem 10 (Preservation). Parallel reduction in DC preserves typing:

\[
\Gamma \vdash t : A \mid \Delta \text{ and } t \doteq u \implies \Gamma \vdash u : A \mid \Delta \\
\Gamma \mid k : A \vdash \Delta \text{ and } k \doteq l \implies \Gamma \mid l : A \vdash \Delta \\
\Gamma \vdash c \vdash \Delta \text{ and } c \doteq d \implies \Gamma \vdash d \vdash \Delta
\]

Proof. By induction on the definition of parallel reduction.

The base case is, as usual, given by preservation for head reduction (theorem 9). The remaining cases are borne by the coupling of remark 1 and the induction hypothesis. The
only more involved cases are those of quantifications due to the side-conditions in their rules.

\[ \Gamma \vdash a(t) : \forall X . F(X) \mid \Delta \text{ and } t \rightsquigarrow u \quad \Rightarrow \quad \Gamma \mid a[k] : \forall X . F(X) \vdash \Delta \text{ and } k \rightsquigarrow l \]

\[ \Gamma \vdash t : F(X) \mid \Delta \text{ and } X \text{ fresh and } t \rightsquigarrow u \quad \Rightarrow \quad \Gamma \mid k : F(T) \vdash \Delta \text{ and } k \rightsquigarrow l \]

\[ \Gamma \vdash u : F(X) \mid \Delta \text{ and } X \text{ fresh} \quad \Rightarrow \quad \Gamma \mid l : F(X) \vdash \Delta \]

\[ \Gamma \vdash e(t) : \exists X . F(X) \mid \Delta \text{ and } t \rightsquigarrow u \quad \Rightarrow \quad \Gamma \mid e[k] : \exists X . F(X) \vdash \Delta \text{ and } k \rightsquigarrow l \]

\[ \Gamma \vdash t : F(T) \mid \Delta \text{ and } t \rightsquigarrow u \quad \Rightarrow \quad \Gamma \mid k : F(X) \vdash \Delta \text{ and } X \text{ fresh and } k \rightsquigarrow l \]

\[ \Gamma \mid u : F(T) \mid \Delta \quad \Rightarrow \quad \Gamma \mid l : F(X) \vdash \Delta \text{ and } X \text{ fresh} \]

\[ \Gamma \vdash e(u) : \exists X . F(X) \mid \Delta \quad \Rightarrow \quad \Gamma \mid e[l] : \exists X . F(X) \vdash \Delta \]

\[ \square \]

3.5 Strong Normalization

§3.14. Strong Normalization for Classical Languages Our goal will now become that of proving that the extension of the Dual Calculus with implicational, subtractive and second-order types is strongly normalizing—i.e. that there are no infinite reduction sequences starting from well-typed phrases. When it comes to Classical Languages, we find two main kinds of proofs: by embedding; and by reducibility.

The first consists in interpreting the source language into some target language that has already been shown to be strongly normalizing. The translation needs to reflect reduction so that any hypothetical infinite chain of reductions would (contradictorily) give rise to an infinite chain of reductions in the target language. The original presentation of the Dual Calculus (Wadler, 2003) came with continuation-passing style translations into the call-by-name and call-by-value simply-typed lambda calculus with co-products—known to be strongly normalizing—which entails as a corollary the strong normalization of DC. Kimura and Tatsuta (2009) proved the strong normalization of the second-order system via an embedding into the second-order symmetric lambda calculus; and then, cunningly, proved the strong normalization of their extension of DC with inductive types by translating it into the second-order Dual Calculus. Outside of the Dual Calculus, Parigot (1997) showed the strong normalization of his second-order \( \lambda \mu \) calculus via an embedding into System F.

Reducibility (Tait, 1967; Girard, 1972; Girard et al., 1989) is the standard model-theoretic technique for proving strong normalization of functional languages and has, naturally, been extended to the Classical setting, too. For propositional languages, the basic ideas are due to Barbanera and Berardi (1996) and their symmetric candidates technique—viz. the completion operation used to handle non-determinism, and the restriction operation used to define the action of abstractions. Dougherty et al. (2005) and Tzevelekos (2006) have adapted this to the Dual Calculus. Parigot (2000) showed how to expand to a full reducibility proof of strong normalization for a second order classical
language. Parigot (1997) also gave a reducibility proof for his $\lambda\mu$ calculus—although, given the more functional nature of the language, the definitions involved are much more similar to the familiar functional case.

§3.15. Structure of the Proof The proof herein descends directly from that of Parigot (2000) but we put more structure into our “reducibility candidates”. We introduce the concept of an orthogonal pair. Their fundamental property is that they are comprised of sets of terms and co-terms which are orthogonal: all the cuts we can construct from those sets are strongly normalizing. Further, these sets will be saturated: they are closed under parallel reduction. We will show that this tweaked formulation can be endowed with the structure of a complete lattice. It will be from this lattice structure that we will later find the fix-points necessary for interpreting induction and co-induction (cf. section 6.5.2). The exact interpretation of types as orthogonal pairs follows. This interpretation needs to be contextualized by the interpretation of the free type variables in a type. The interpretation preserves substitution of types and also the weakening of the interpretation context. From these properties it will be possible to describe how constructors and eliminators preserve type interpretations; and thence we derive adequacy of our model: any well-typed phrase of the Dual Calculus is strongly normalizing for any substitution that respects the interpretation—like the identity substitution.

3.5.1 Sets and Saturation

§3.16. Sets of Syntax Given the preponderance of sets to our analysis, we begin by showing how various constructions on the phrases of the Dual Calculus can be generalised to sets of phrases. The syntax forming rules of DC and its extensions determine three sets: the set $T$ of terms, $K$ of co-terms, and $C$ of cuts. One very important subset of $C$ is the set $SN$ of strongly normalizing cuts—those cuts $c$ for which we cannot construct an infinite reduction sequence 

$$c \Rightarrow c_1 \Rightarrow c_2 \Rightarrow \ldots \Rightarrow c_n \Rightarrow \ldots \quad (n \in \mathbb{N})$$

Likewise, the sets $SN\pi \subseteq T$ and $SN\mathcal{K} \subseteq K$ contain all those terms which are strongly normalizing under the congruent closure of head reduction. The set $IT \subseteq T$ shall denote the set of terms that represent introductions:

$$\langle t, t' \rangle, t_1(t), t_2(t), not(k), \lambda x(t), (t \# k), a(t), e(t)$$

conversely, the set $E\mathcal{K}$ shall denote the set of co-terms of the form:

$$fst[k], snd[k], [k, k'], not[k], (t @ k), \mu \alpha(k), a[k], e[k]$$

§3.17. Images The syntactic operators of DC are mappings of phrases to phrases; reduction is also defined in terms of individual phrases. As we move towards a more set based view of the language, we need to generalise those constructions to sets of (suitable) phrases. The way to do it is by taking the image construction. In its full generality, $[R](A)$, the image of a set $A$ under relation $R$, is the set

$$\{ x \in \text{cod}(R) \mid \text{for some } a \in A, aRx \}.$$
The image of any relation is monotone (with respect to the argument set) and it preserves arbitrary unions—facts we shall make much (implicit) use of later on. For any function \( f : A \rightarrow B \), the above definition simplifies to:

\[
[f](A) = \{ f(a) \mid a \in A \}.
\]

§3.18. Syntactic Actions  For constructors and eliminators, our first step is to create actions that do for sets of terms and co-terms what the operators of the calculus do for individual phrases—we shall term these syntactic actions (table 3.3). They are simply the image of the syntactic operators seen as functions of from phrases to phrases (of appropriate class). By abuse of notation, we represent these images using the same notation as the underlying operators.

§3.19. Reduction  Reduction at the level of sets is, again, just the image of the reduction relation. Paramount to strong reduction, we give explicit formulas for its calculation in table 3.4. A negation term reduces iff its inner continuation reduces; on the set level we have that reductions of \( \text{not}(K) \) reflect exactly the reductions in \( K \). In a pairing we have a reduction whenever either of the components reduce—leaving the other component unaltered. The set of possible reductions from a pair is formed by the union of these two alternatives. One final, and the most complicated, example is that of cuts. Here we have three alternatives: either by a parallel reduction on its term or co-term side; or, the cut itself reduces via a head reduction.

§3.20. Saturation and Image of Reduction  Given a set of co-terms \( K \subseteq \mathcal{K} \), we will often ask if it is closed under reduction:

\[
k \in K \text{ and } k \leadsto l \implies l \in K.
\]

Set \( K \) will then be said to be saturated (under reduction). Saturation can be defined for sets of terms and cuts in a similar manner. We can rephrase this definition using the image of parallel reduction.

**Proposition 5.** A set of phrases \( P \) is saturated iff

\[
\left[\rightsquigarrow\right](P) \subseteq P
\]

**Proof.** Assume that for any \( t \in P \) if \( t \leadsto u \) then \( u \in P \); then

\[
u \in \left[\rightsquigarrow\right] P \implies \text{for some } t \in P, \ t \leadsto u \implies u \in P
\]

For the converse, assume \( \left[\rightsquigarrow\right](P) \subseteq P \) and reason:

\[
t \leadsto u \text{ and } t \in P \implies u \in \left[\rightsquigarrow\right](P) \implies u \in P
\]
### Reduction Properties of the Dual Calculus

\[
\begin{align*}
\langle -, - \rangle : (\mathcal{P}(T))^2 & \to \mathcal{P}(T) \\
\langle T, T' \rangle = \{ \langle t, t' \rangle \mid t \in T, t' \in T' \} & \quad [\langle -, - \rangle : (\mathcal{P}(K))^2 \to \mathcal{P}(K)] \\
\text{fst}[\cdot] : \mathcal{P}(K) & \to \mathcal{P}(K) \quad \text{fst}[K] = \{ \text{fst}[k] \mid k \in K \} \\
\text{snd}[\cdot] : \mathcal{P}(K) & \to \mathcal{P}(K) \quad \text{snd}[K] = \{ \text{snd}[k] \mid k \in K \} \\
\text{not}(\cdot) : \mathcal{P}(K) & \to \mathcal{P}(T) \quad \text{not}(K) = \{ \text{not}[k] \mid k \in K \} \\
\lambda x.(T) & = \{ \lambda x. (t) \mid t \in T \} \quad \mu \alpha.(K) = \{ \mu \alpha.(k) \mid k \in K \} \\
(\_ \oplus \_)[K] & = \{ (t \oplus k) \mid t \in T, k \in K \} \\
\text{add}(\cdot) & : \mathcal{P}(T) \to \mathcal{P}(T) \quad \text{add}(\cdot)[K] = \{ \text{add}(k) \mid k \in K \} \\
\text{send}(\cdot) & : \mathcal{P}(T) \to \mathcal{P}(T) \quad \text{send}(\cdot)[K] = \{ \text{send}(k) \mid k \in K \} \\
\text{eval}(\cdot) & : \mathcal{P}(T) \to \mathcal{T} \quad \text{eval}(\cdot)[K] = \{ \text{eval}(k) \mid k \in K \} \\
\_ \otimes \_ & : \mathcal{P}(T) \times \mathcal{P}(K) \to \mathcal{P}(\mathcal{T}) \\
T \otimes K & = \{ t \otimes k \mid t \in T, k \in K \} \\
\_ \odot \_ & : \mathcal{P}(\mathcal{V}) \times \mathcal{P}(\mathcal{C}) \to \mathcal{P}(\mathcal{T}) \\
\_ \odot \_ & : \mathcal{P}(\mathcal{C}) \to \mathcal{P}(\mathcal{C}) \\
\alpha.(\_)[C] & = \{ \alpha.(c) \mid c \in C \} \\
\_ & : \mathcal{P}(\mathcal{C}) \to \mathcal{P}(\mathcal{C})
\end{align*}
\]

Table 3.3: Syntactic actions for second-order DC
STRONG NORMALIZATION

\[ \text{[\text{\[\text{id}\]}]}(u) = \langle \text{[\text{id}}(u) \rangle \]
\[ \text{[\text{\[\text{fst}\]}]}(K) = \text{\[\text{fst}\]}(\text{[\text{id}}(K)) \]
\[ \text{[\text{\[\text{snd}\]}]}(K) = \text{\[\text{snd}\]}(\text{[\text{id}}(K)) \]
\[ \text{[\text{\[\text{fst}\]}]}(T) = \text{\[\text{fst}\]}(\text{[\text{id}}(T)) \]
\[ \text{[\text{\[\text{snd}\]}]}(T) = \text{\[\text{snd}\]}(\text{[\text{id}}(T)) \]
\[ \text{[\text{\[\text{not}\]}]}(K) = \text{\[\text{not}\]}(\text{[\text{id}}(K)) \]
\[ \text{[\text{\[\text{not}\]}]}(T) = \text{\[\text{not}\]}(\text{[\text{id}}(T)) \]

Table 3.4: Formulas for calculation of image of reduction

Remark 2. The sets

\[ \emptyset, \text{Var, Covar, } \mathcal{T}, \mathcal{K}, \mathcal{C}, \mathcal{IT}, \mathcal{E}, \mathcal{SN}, \mathcal{S}|\mathcal{T}, \mathcal{S}|\mathcal{N}, \mathcal{S}|\mathcal{K}, \text{ and } \mathcal{S}|\mathcal{N} \]

are all saturated. The image of the empty set by any relation is the empty set; variables and co-variables do not reduce and, therefore, the image under reduction of their sets is the empty set; reduction for terms and co-terms is defined by homomorphism, so every reduction on an introduction or elimination retains said introduction or elimination; terms, co-terms, and cuts all reduce, respectively, to terms, co-terms, and cuts; and, by virtue of strong normalization, strongly normalizing terms, co-terms and cuts must reduce to other strongly normalizing phrases.

Lemma 5 (Saturation for (Co-) Term Syntactic Operators). Assume \( T \) and \( U \), \( K \) and \( L \), and \( C \) are saturated sets of terms, co-terms, and cuts, respectively. Then the sets constructed from them using the introduction, elimination, and structural abstraction syntactic operators are all saturated:

\[ \text{[\text{\[\text{id}\]}]}(\langle T, U \rangle) \subseteq \langle T, U \rangle \]
\[ \text{[\text{\[\text{fst}\]}]}(K) \subseteq \text{\[\text{fst}\]}(K) \]
\[ \text{[\text{\[\text{snd}\]}]}(K) \subseteq \text{\[\text{snd}\]}(K) \]
\[ \text{[\text{\[\text{fst}\]}]}(T) \subseteq \text{\[\text{fst}\]}(T) \]
\[ \text{[\text{\[\text{snd}\]}]}(T) \subseteq \text{\[\text{snd}\]}(T) \]

67
REDUCTION PROPERTIES OF THE DUAL CALCULUS

\[
\begin{align*}
&\begin{array}{llll}
[K] (\text{not}(K)) \subseteq \text{not}(K) & [K] (\text{not}(T)) \subseteq \text{not}(T) \\
&\vdots \\
&[T] (\lambda x.(T)) \subseteq \lambda x.(T) & [T] ((T \ @ K)) \subseteq (T \ @ K) \\
&[T] ((T \ # K)) \subseteq (T \ # K) & [T] (\mu \alpha.(K)) \subseteq \mu \alpha.(K) \\
&[a(T)] \subseteq a(T) & [a(K)] \subseteq a(K) \\
&[e(T)] \subseteq e(T) & [e(K)] \subseteq e(K) \\
&[\alpha(C)] \subseteq \alpha(C) & [x.(C)] \subseteq x.(C)
\end{array}
\]

\]

Proof. Repeated use of the formulas in table 3.4 together with the (set) monotonicity of the syntactic operations. 

\[\square\]

§3.21. Restriction Some cuts—viz. those comprised of abstractions, both on the structural side of the calculus, and also term and co-term abstractions in implicative and subtractive types—do not simply reduce to cuts made out of sub-phrases of the starting phrases, but to phrases obtained via substitution. To enforce the invariant that those resulting phrases still be well-behaved, we need two additional restriction operations—respectively, on substitution by terms and by co-terms. Take \(T\) to stand for a set of terms, \(K\) for one of co-terms, and \(C\) for cuts; consider them to be—informally—our initial set of well-behaved phrases. Additionally, take sets \(T' \subseteq T\) and \(K' \subseteq K\) to be those terms and co-terms for which substitution may occur for \(x \in \text{Var}\) and \(\alpha \in \text{Covar}\), respectively; the restricted sets of (well-behaved) phrases are then, respectively, the sets

\[
\begin{align*}
T' \frac{v}{x} & = \{ t \in SN_T \mid \forall t' \in T'.t'[t'/x] \in T \} \\
K' \frac{v}{x} & = \{ k \in SN_K \mid \forall t' \in T'.k[t'/x] \in K \} \\
C' \frac{v}{x} & = \{ c \in SN \mid \forall t' \in T'.c[t'/x] \in C \}
\end{align*}
\]

Restriction and substitution are related by a universal property.

Proposition 6. Let \(U \subseteq T\) (resp. \(L \subseteq K\)) and \(x \in \text{Var}\) (resp. \(\alpha \in \text{Covar}\)). The restriction is right adjoint to point-wise substitution:

\[
\begin{align*}
&T \frac{U}{x} \subseteq V \iff T \subseteq V \frac{U}{x} & -[U]_x : T \rightarrow SN_T \\
&T \frac{L}{\alpha} \subseteq V \iff T \subseteq V \frac{L}{\alpha} & -[L]_\alpha : T \rightarrow SN_K \\
&K \frac{U}{x} \subseteq V \iff K \subseteq V \frac{U}{x} & -[U]_x : K \rightarrow SN_K \\
&K \frac{L}{\alpha} \subseteq V \iff K \subseteq V \frac{L}{\alpha} & -[L]_\alpha : K \rightarrow SN_K \\
&C \frac{U}{x} \subseteq V \iff C \subseteq V \frac{U}{x} & -[U]_x : C \rightarrow SN \\
&C \frac{L}{\alpha} \subseteq V \iff C \subseteq V \frac{L}{\alpha} & -[L]_\alpha : C \rightarrow SN
\end{align*}
\]
STRONG NORMALIZATION

Proof. Assume $T[U/x] \subseteq V$;

$$
t \in T \implies \text{for any } u \in U, t[u/x] \in T[U/x], t \in SN\cT \quad \text{(subs. pointwise)}
$$

$$
\implies \text{for any } u \in U, t[u/x] \in V, t \in SN\cT \quad \text{(assump.)}
$$

$$
\implies t \in V[U]_x \quad \text{(def.)}
$$

Assume that $T \subseteq V[U]_x$;

$$
v \in T[U/x] \implies \text{for some } t \in T \text{ and } u \in U, v = t[u/x]
$$

$$
\implies \text{for some } t \in V[U]_x, v = t[u/x]
$$

$$
\implies v = t[u/x] \in V
$$

$\square$

§3.22. Substitutivity  The final property that we shall need to lift to the level of sets of syntax is substitutivity—viz. the property that reduction preserves substitution. As a consequence of substitutivity, we will be able to derive the preservation of saturation by the restriction operations.

**Proposition 7.** Let $P$ stand for a set of terms, or of co-terms, or cuts; and $U$ be a set of terms and $L$ a set of co-terms. Substitutivity (I, theorem 7) is equivalent to

$$
([\equiv] (P))[U/x] \subseteq [\equiv] (P[U/x]) \quad ([\equiv] (L)[\alpha] \subseteq [\equiv] (P[L/\alpha])
$$

Proof. Assume substitutivity (theorem 7);

$$
q \in ([\equiv] (P))[U/x] \implies \text{for some } p' \in [\equiv] (P) \text{ and } u \in U, q = p'[u/x]
$$

$$
\implies \text{for some } p \in P \text{ and } u \in U, p \equiv p' \text{ and } q = p'[u/x]
$$

$$
\implies \text{for some } p \in P \text{ and } u \in U, p[u/x] \equiv p'[u/x] = q
$$

$$
\implies q \in [\equiv] (P[U/x])
$$

In the other direction we have

$$
p \equiv p' \implies p' \in [\equiv] (p)
$$

$$
\implies p'[u/x] \in [\equiv] ([p])[u/x]
$$

$$
\implies p'[u/x] \in [\equiv] ((p)[u/x])
$$

$$
\implies p'[u/x] \in [\equiv] ([p[u/x]])
$$

$$
\implies \text{for some } q \in \{p[u/x]\}, q \equiv p'[u/x]
$$

$$
\implies p[u/x] \equiv p'[u/x]
$$

Proof for substitution of co-variables is exactly the same.  $\square$

69
Theorem 11. Let $P$ be a saturated set of either terms, co-terms, or cuts. Then, for any $T \subseteq \mathcal{T}$ and any $K \subseteq \mathcal{K}$, the sets $P|_T$ and $P|_K$ are saturated.

Proof. For either case, a simple algebraic derivation—using the adjointness of the restrictions, monotonicity of the image of a relation, substitutivity, and saturation—suffices.

3.5.2 Orthogonal Pairs

§3.23. Lattices of Syntax

Having studied a few sets relevant to our pursuit of strong normalization of the Dual Calculus, we can now go up a level and add more structure in the form of lattices. The powersets—sets of subsets—$\mathcal{P}(\mathcal{T}), \mathcal{P}(\mathcal{K})$, and $\mathcal{P}(C)$, of the sets of terms, co-terms, and cuts (resp.), each form complete lattices when we take the order to be set inclusion $\subseteq$: the bottom element $\bot$ is always the empty set; the top elements $\top$ are the sets $\mathcal{T}, \mathcal{K}$, and $C$ themselves; and, we can also take arbitrary unions and intersections of elements of $\mathcal{P}(\mathcal{T}), \mathcal{P}(\mathcal{K})$, and $\mathcal{P}(C)$ and they satisfy the least-upper/greatest-lower bound properties—e.g., for any $S \subseteq \mathcal{P}(\mathcal{T})$, its least upper bound (or supremum, or lub for short) is

$$\bigvee S = \bigcup_{T \in S} T \in \mathcal{P}(\mathcal{T}),$$

and the greatest lower bound (or infimum, or glb for short) is

$$\bigwedge S = \bigcap_{T \in S} T \in \mathcal{P}(\mathcal{T}).$$

Similarly, the lattices of sets of strongly normalizing phrases, $\mathcal{P}(SN\mathcal{T}), \mathcal{P}(SN\mathcal{K})$, and $\mathcal{P}(SN)$ are all complete lattices.

§3.24. Combinations of Lattices

There are many different ways to combine lattices to create new ones (cf. [Davey and Priestley, 2002, 2.14]). In the sequel we will make extensive use of two. The first is the dualization $L^*$ of a (complete) lattice $L$. Here we take exactly the elements $L$ but consider them in the opposite order. Naturally, we must interchange lubs and glbs, and top and bottom elements. The second operation is the

\[\text{For this and other assumptions and terminology on sets and lattices see the book by } \text{Davey and Priestley (2002).}\]
STRONG NORMALIZATION

product \( L \times L' \) of (complete) lattices \( L \) and \( L' \); here, the order and the operations are taken pointwise—e.g.

\[
\bot_{L \times L'} = (\bot_L, \bot_{L'}) \quad (a, a') \leq_{L \times L'} (b, b') \text{ iff } a \leq_L b \text{ and } a' \leq_{L'} b'.
\]

In both cases, if the underlying lattices are complete then so is the resulting one.

§3.25. Orthogonal Pairs We can also go in a different direction and take a lattice and restrict it to only certain elements of the carrier sets. Given a pair \( (T, K) \in \mathcal{P}(SN\mathcal{T}) \times \mathcal{P}^\ast(SN\mathcal{K}) \) of sets of strongly normalizing terms/co-terms, we call it an orthogonal pair iff

\[
T \cdot K \subseteq SN^\preceq \quad \text{and} \quad [\preceq](T) \subseteq T, \quad [\preceq](K) \subseteq K
\]
i.e. if its components are saturated and if all the cuts we can construct from them are strongly normalizing. The first condition is known as the orthogonality condition proper, and the other two, naturally, as the saturation conditions.\(^4\)

Given an orthogonal pair \( P = (T, K) \), we will often need to address its components individually; the abbreviations \((P)\up^\top = T\) for the term part, and \((P)\up^K = K\) for the co-term part shall be quite useful. Straightaway, a few examples spring to mind: the pair \((\text{Var}, \text{Covar})\) is orthogonal (no reductions are possible); even simpler, the pair \((\emptyset, \emptyset)\) is trivially orthogonal! In fact, \((\emptyset, SN\mathcal{K})\) and \((SN\mathcal{T}, \emptyset)\) are both orthogonal—as trivial sets render the orthogonal condition trivial and the sets of strongly normalising terms and of strongly normalizing co-terms are saturated. Orthogonal pairs form a (complete) sub-lattice \( \mathcal{OP} \) of the lattice \( \mathcal{P}(SN\mathcal{T}) \times \mathcal{P}^\ast(SN\mathcal{K}) \)—i.e. they preserve the order and (non-empty) lubs and glbs that exist in the latter. The full lattice structure is spelt out in the following proposition. Note how the differing variances of the order are paramount: as we grow one side of the pair, we are forced to shrink the other side to maintain orthogonality by removing everything that is not orthogonal to the additional elements.

**Proposition 8** (Lattice Structure of \( \mathcal{OP} \)). The set of orthogonal pairs is a sub-lattice of the lattice \( \mathcal{P}(SN\mathcal{T}) \times \mathcal{P}^\ast(SN\mathcal{K}) \); i.e., for \( P, Q \in \mathcal{OP} \),

\[
P \leq Q \text{ iff } (P)\up^\top \subseteq (Q)\up^\top \text{ and } (P)\up^K \supseteq (Q)\up^K;
\]

the join and meet of arbitrary non-empty sets \( S \subseteq \mathcal{OP} \) is

\[
\bigvee S \triangleq \left( \bigcup_{P \in S} (P)\up^\top, \bigcap_{P \in S} (P)\up^K \right) \quad \text{and} \quad \bigwedge S \triangleq \left( \bigcap_{P \in S} (P)\up^\top, \bigcup_{P \in S} (P)\up^K \right),
\]

and the empty joins and meets—the extrema—are

\[
\bot \triangleq (\emptyset, SN\mathcal{K}) \quad \text{and} \quad \top \triangleq (SN\mathcal{T}, \emptyset).
\]

\(^4\) Given that orthogonal pairs are defined using an orthogonality condition and the two saturation conditions one may ask, then, why we call these pairs simply orthogonal. We chose this nomenclature because the generalisation to saturation properties corresponds to a necessary weakening of the induction hypothesis implicit in the establishment of orthogonality—it turns out that the orthogonality condition alone is not enough (in general) to inductively prove orthogonality for the sets constructed via realizability interpretations (Girard et al., 1989; Gallier, 1989).
Proof. Starting with the extrema, they coincide with the ones from the super-lattice, so we are guaranteed that they are extrema for the inherited order. We have alluded before to their saturation (remark 2); we need only to confirm that they are orthogonal:

\[ \emptyset \cdot S \cap \mathcal{K} = S \cap \mathcal{T} \cdot \emptyset = \emptyset \subseteq S \mathcal{N}. \]

Now, let \( S \subseteq \mathcal{O} \mathcal{P} \) be non-empty; we have that

\[ t \in \left( \bigvee_{P \in S} P \right)^T \text{ and } k \in \left( \bigvee_{P \in S} P \right)^K \]

iff \( t \in \bigcup_{P \in S} (P)^T \text{ and } k \in \bigcap_{P \in S} (P)^K \)

iff \( t \in (P)^T \), for some \( P \in S \); and, for all \( P \in S \), \( k \in (P)^K \)

\( \Rightarrow \) for some \( P \in S \), \( t \in (P)^T \) and \( k \in (P)^K \)

\( \Rightarrow \ t \cdot k \in S \mathcal{N} \)

(this last step following by our assumption on the orthogonality of the elements of \( S \)). We have just proven that the components of our purported join are orthogonal; for saturation we have that, by saturation of each component of an orthogonal pair

for any \( P \in S \), \( \bigvee (P)^T \subseteq (P)^T \)

\( \Rightarrow \) for any \( P \in S \), \( \bigvee (P)^T \subseteq \bigcup_{P \in S} (P)^T \)

\( \Rightarrow \bigcup_{P \in S} \bigvee (P)^T \subseteq \bigcup_{P \in S} (P)^T \)

\( \Rightarrow \bigvee \left( \bigvee S \right)^T \subseteq \left( \bigvee S \right)^T \)

and

for any \( P \in S \), \( \bigvee (P)^K \subseteq (P)^K \)

\( \Rightarrow \) for any \( P \in S \), \( \bigvee (P)^K \subseteq \bigcap_{P \in S} (P)^K \)

\( \Rightarrow \bigcap_{P \in S} (P)^K \subseteq \bigcap_{P \in S} (P)^K \)

\( \Rightarrow \bigvee (S)^K \subseteq (S)^K \)

Minimality w.r.t. the given order holds because the joins are inherited from the super-lattice.

The proof for meets is simply the dual. \( \square \)
§3.26. Type Structure—or Lack Thereof  It is important to note that orthogonal pairs need not respect the type structure. For example, under this definition of orthogonality, pairings \( \langle x, y \rangle \) are orthogonal to any disjunction continuation such as \([a, \beta]\)—as no reduction rule applies—and, so, both can coexist in an orthogonal pair. In a sense, the sensible orthogonal pairs that interest us live in a world of strange pairings of sets that are, nonetheless, sound when it comes to strong normalization.

§3.27. Orthogonal Normal Pairs  Our next object of interest will be those orthogonal pairs that are made out solely of constructors on the term side and eliminators on the co-term side. These are interesting—indeed vital for our later undertakings—because they are the “values” of types, as opposed to variables and abstractions which exist only to implement computational behaviour in the calculus. The lattice of orthogonal normal pairs \( \mathcal{ONP} \) is the sub-lattice of elements \( N \) of \( \mathcal{OP} \) for which

\[
(N)^T \subseteq IT \quad \text{and} \quad (N)^K \subseteq E\mathcal{K}.
\]

**Proposition 9** (Lattice Structure of \( \mathcal{ONP} \)). The lattice \( \mathcal{ONP} \) is a sub-lattice of \( \mathcal{OP} \); it is also complete with extrema given by

\[
\bot = (\emptyset, E\mathcal{K} \cap S\mathcal{NK}) \quad \text{and} \quad \top = (IT \cap S\mathcal{NT}, \emptyset).
\]

**Proof.** Take \( N \in \mathcal{ONP} \). By the definition of \( \mathcal{OP} \) and \( \mathcal{ONP} \), respectively, it must satisfy

\[
(\bot)^T = \emptyset \subseteq (N)^T \subseteq S\mathcal{NT} \quad \text{and} \quad (N)^T \subseteq IT
\]

whence, also

\[
(N)^T \subseteq IT \cap S\mathcal{NT} = (\top)^T;
\]

similarly,

\[
(\bot)^K = E\mathcal{K} \cap S\mathcal{NK} \supseteq (N)^K \supseteq \emptyset = (\top)^K.
\]

Orthogonality of these is trivial. For saturation, we know that \( IT, E\mathcal{K}, S\mathcal{NT} \) and \( S\mathcal{NK} \) are saturated, and so is the intersection of any combination of them.

Now take a non-empty \( S \subseteq \mathcal{ONP} \); we need only to prove that its join and meet satisfy the additional property of normal pairs. For any \( N \in S \)

\[
(N)^T \subseteq IT \cap S\mathcal{NT} \quad \text{and} \quad (N)^K \subseteq E\mathcal{K} \cap S\mathcal{NK};
\]

hence,

\[
\bigcup_{N \in S} (N)^T \subseteq IT \cap S\mathcal{NT} \quad \text{and} \quad \bigcup_{N \in S} (N)^K \subseteq E\mathcal{K} \cap S\mathcal{NK};
\]

and also

\[
\bigcap_{N \in S} (N)^T \subseteq IT \cap S\mathcal{NT} \quad \text{and} \quad \bigcap_{N \in S} (N)^K \subseteq E\mathcal{K} \cap S\mathcal{NK}.
\]

Their components in \( IT \) and \( E\mathcal{K} \), we conclude that the joins and meets are in \( \mathcal{ONP} \). \( \square \)
3.5.3 Orthogonal Actions

§3.28. “Types” for Orthogonal Pairs  This section and the next deal with how to go from one lattice to the other—how to construct orthogonal normal pairs from orthogonal pairs, and vice-versa—and how to do this in a way that is true to the typed structure of the calculus.

The distinction to keep in mind is that of our (intuitive) values and arbitrary terms. This section will deal with how to create orthogonal normal pairs that encompass the actions of the introductions/eliminations of any type on the set level. The next section shows how—relative to these values—we can complete (operation \(\parallel\)) the picture with all compatible (co-) variables and (co-) terms into one orthogonal pair. As the pairs need to be shown orthogonal, we will, first, give a sufficient condition for orthogonality that is based solely on head reduction—and saturation, here, is essential.

Lemma 6 (Saturation and Orthogonality). Let \(T \subseteq SN T\) and \(K \subseteq SN K\) be saturated sets of terms and co-terms, respectively, satisfying \([\rightsquigarrow](T \cdot K) \subseteq SN\); then \(T \cdot K \subseteq SN\).

Proof. We prove the result indirectly by proving instead that for any finite subsets \(T' \subseteq T\) and \(K' \subseteq K\), \(T' \cdot K' \subseteq SN\). Given that both \(T\) and \(K\) are equal to the union of their finite subsets (and these are preserved by the syntactic operation \(- \cdot -\)), the result will follow. The proof for these will be by induction on the sum of the depths of all possible reduction paths of all phrases in the sets \(T'\) and \(K'\). As the sets are finite and the (co-) terms terminating (by virtue of being in \(SN T\) or \(SN K\), as appropriate) we are guaranteed that this measure is either zero—and only head reductions apply—or strictly reducing when we execute a single step of parallel reduction.

For the zero case we have that

\[
[SN T] (T') \cdot K' = \emptyset \subseteq SN'\quad \text{and}\quad T' \cdot (SN K) = \emptyset \subseteq SN'
\]

\[
\Rightarrow [\rightsquigarrow] (T' \cdot K') \subseteq SN'\quad \text{and}\quad (\exists S) (T') \cdot K' \subseteq SN'\quad \text{and}\quad T' \cdot (SN K) \subseteq SN'
\]

\[
\Rightarrow [\rightsquigarrow] (T' \cdot K') \cup (\exists S) (T') \cdot K' \cup T' \cdot (\exists S) (K') \subseteq SN'
\]

\[
\Rightarrow \exists S (T' \cdot K') \subseteq SN'
\]

\[
\Rightarrow T' \cdot K' \subseteq SN
\]

The induction step is proved with measured aid of the induction hypothesis. Assume the sum of the depths of all reduction paths for terms and co-terms in \(T'\) and \(K'\) is non-zero; it therefore follows that either

\[
(\exists S) (T') \cdot K'\quad \text{and}\quad T' \cdot (\exists S) (K')
\]

both have smaller sum than \(T' \cdot K'\), or the sum of depths of reduction for one of them is zero and for the other it is smaller that that of \(T' \cdot K'\). Saturation now takes on a critical role: because of it we know that both \((\exists S) (T')\) and \((\exists S) (K')\) are also finite subsets of
STRONG NORMALIZATION

$T$ and $K$, and, therefore the induction hypothesis can be applied to them. Therefore, in any case of the cases above, be it by the induction hypothesis or by virtue of not having any reducible terms, we have that

$$\left(\left[\rightarrow\right](T')\right) \cdot K' \subseteq SN \quad \text{and} \quad T' \cdot \left(\left[\rightarrow\right](K')\right) \subseteq SN;$$

and, therefore,

$$\left[\rightarrow\right](T \cdot K) \subseteq SN \quad \text{and} \quad \left[\rightarrow\right](T' \cdot \left(\left[\rightarrow\right](K')\right)) \subseteq SN
\implies \left[\rightarrow\right](T' \cdot K') \subseteq SN \quad \text{and} \quad \left[\rightarrow\right](\left[\rightarrow\right](T')) \cdot K' \subseteq SN
\implies \left[\rightarrow\right](T' \cdot K') \subseteq SN \quad \text{and} \quad \left[\rightarrow\right](\left[\rightarrow\right](T')) \cdot K' \subseteq SN
\implies \left[\rightarrow\right](\left[\rightarrow\right](T' \cdot K')) \subseteq SN
\implies T' \cdot K' \subseteq SN$$

\[\square\]

**Lemma 7** (Preservation of Orthogonality). *Let $T, T' \subseteq SN^T$, $K, K' \subseteq SN^K$ be saturated sets such that

$T \cdot K \subseteq SN$ and $T' \cdot K' \subseteq SN$;

then:

$$\langle T, T' \rangle \cdot \text{fst}[K] \subseteq SN$$
$$i_1(T) \cdot \left[\left[\rightarrow\right](K, K')\right] \subseteq SN$$
$$\lambda x. (T'[\bar{x}]) \cdot (T @ K') \subseteq SN$$
$$\mu a. \left(\left[\rightarrow\right](K_{a'})\right) \subseteq SN$$
$$\langle T, T' \rangle \cdot \text{snd}[K'] \subseteq SN$$
$$i_2\langle T' \rangle \cdot \left[\left[\rightarrow\right](K, K')\right] \subseteq SN$$
$$a(T') \cdot a[K] \subseteq SN$$
$$\langle T, T' \rangle \cdot \text{not}[T] \subseteq SN$$
$$\text{not}(K) \cdot \text{not}[T] \subseteq SN$$
$$e(T) \cdot e[K] \subseteq SN$$
$$(T \# K') \cdot \left[\left[\rightarrow\right](K, K')\right] \subseteq SN$$

**Proof.** By lemma 6, it suffices to show that the head reducts of each case are in $SN$ as, by lemma 5, all the sets in the cuts are saturated and quite obviously in $SN^T$ or $SN^K$, as appropriate.

$$\left[\rightarrow\right]\left(\left[\rightarrow\right]\left(\left[\rightarrow\right](T, T') \cdot \text{fst}[K]\right)\right) = T \cdot K \subseteq SN$$
$$\left[\rightarrow\right]\left(\left[\rightarrow\right]\left(\left[\rightarrow\right](T, T') \cdot \text{snd}[K']\right)\right) = T' \cdot K' \subseteq SN$$
$$\left[\rightarrow\right]\left(\left[\rightarrow\right]\left(\left[\rightarrow\right](i_1(T) \cdot \left[\left[\rightarrow\right](K, K')\right]\right)\right)\right) = T \cdot K \subseteq SN$$
$$\left[\rightarrow\right]\left(\left[\rightarrow\right]\left(\left[\rightarrow\right](i_2\langle T' \rangle \cdot \left[\left[\rightarrow\right](K, K')\right]\right)\right)\right) = T' \cdot K' \subseteq SN$$
$$\left[\rightarrow\right]\left(\left[\rightarrow\right]\left(\left[\rightarrow\right](\lambda x. (T'[\bar{x}]) \cdot (T @ K')\right)\right)\right) = T' \cdot K' \subseteq SN$$
$$\left[\rightarrow\right]\left(\left[\rightarrow\right]\left(\left[\rightarrow\right](\mu a. \left(\left[\rightarrow\right](K_{a'})\right))\right)\right) = T' \cdot K' \subseteq SN$$
$$\left[\rightarrow\right]\left(\left[\rightarrow\right]\left(\left[\rightarrow\right](\langle T, T' \rangle \cdot \text{not}[T])\right)\right) = T \cdot K \subseteq SN$$
$$\left[\rightarrow\right]\left(\left[\rightarrow\right]\left(\left[\rightarrow\right](a(T) \cdot a[K])\right)\right) = T \cdot K \subseteq SN$$
$$\left[\rightarrow\right]\left(\left[\rightarrow\right]\left(\left[\rightarrow\right](e(T) \cdot e[K])\right)\right) = T \cdot K \subseteq SN$$

For implicational and subtractive abstractions we note, first, that because $T'$ (respectively $K$) is saturated, then so is $T'[\bar{x}]$ (respectively $K_{a'}$) by theorem 11; then, again by lemma 5.
we have that both sides of each cut are saturated; we prove, then, for arbitrary $x \in \text{Var}$, $\alpha \in \text{Covar}$, and with the aid of the adjointness of substitution/restriction that

$$[\to](\lambda x. (T'|_x^T \cdot (T @ K'))) = (T'|_x^T [T/x]) \cdot K' \subseteq T' \cdot K' \subseteq SN$$

$$[\to](T \# K') \cdot (K|_a^K') = T \cdot (K|_a^K' [K'/a]) \subseteq T \cdot K \subseteq SN$$

# §3.29. Type Actions

At last, we can sensibly bring the action of the logical connectives and quantifiers to the level of orthogonal normal pairs. The idea is to use these actions to determine what the normal terms and co-terms of a type are. They are parametrised by orthogonal pairs—rather than normal pairs—for we consider normality to be defined solely by the outermost syntactic operator. The complete definition of the actions can be found in table 3.5 for $P, Q \in \mathcal{OP}$ and $S \subseteq \mathcal{OP}$.

For example, the conjunctive action for two orthogonal pairs $P$ and $Q$ takes the set of (DC) pairs of their terms, and the projections that can be sensibly constructed from their co-terms:

$$P \land Q = \bigvee_{x \in \text{Var}} (\lambda x. (Q|_x^T), (P|_x^T) : (P|_x^T @ (Q|_x^T)))$$

We can easily see that this is actually the greatest lower bound of two elements of $\mathcal{ON}\mathcal{P}$,

$$\bigvee_{x \in \text{Var}} (\lambda x. (Q|_x^T), (P|_x^T)) : \bigvee_{x \in \text{Var}} (\lambda x. (P|_x^T @ (Q|_x^T)))$$

one for each projection, and thus the orthogonality follows from the orthogonality of each component.

For functional types, special care needs to be taken with the abstractions. For an implication $P \rightarrow Q$, the set of valid abstractions are those constructed from terms whose substitution for arbitrary terms of $P$ is also in $Q$.

$$P \rightarrow Q = \bigvee_{x \in \text{Var}} (\lambda x. (Q|_x^T), (P|_x^T @ (Q|_x^T)))$$

$$= \bigvee_{x \in \text{Var}} (\lambda x. (Q|_x^T), (P|_x^T) @ (Q|_x^T))$$

For universal quantifications, we are interested in the terms that are common to every particular instance of a typescheme. To understand the universal quantification action, we look at a parameter set $S$ as containing exactly the interpretations of each of those instances; we then add the universal quantification operators and use the lattice structure of $\mathcal{ON}\mathcal{P}$—the meet—to get only and all the common terms (and union of continuations). In the existential case we do the dual, taking joins to form the base operational sets.

$$\forall S = \bigwedge_{P \in S} (a((P|_x^T), a[(P|_x^T)])$$

As expected these operations yield orthogonal normal pairs whenever applied to orthogonal pairs.
STRONG NORMALIZATION

\[ P \land Q = (\langle (P)^\uparrow, (Q)^\uparrow \rangle, \text{fst}[\left(P\right)^K] \cup \text{snd}[\left(Q\right)^K]) \]
\[ P \lor Q = (t_1\langle (P)^\uparrow \rangle \cup t_2\langle (Q)^\uparrow \rangle, \left[(P)^K, (Q)^K\right]) \]
\[ \neg P = (\text{not}\langle (P)^K \rangle, \text{not}[\left(P\right)^T]) \]
\[ P \rightarrow Q = \bigvee_{x \in \text{Var}} \left(\lambda x.\left(\left((Q)^\uparrow\right)^{\left(P\right)^x}\right), \left((P)^\uparrow \circ \left(Q\right)^K\right)^{\left(P\right)^x}\right) \]
\[ P \setminus Q = \bigwedge_{a \in \text{Covar}} \left(\left((P)^\uparrow \#(Q)^K\right)^{\left[P\right]^x}, \mu a.\left((P)^K\right)^{\left(Q\right)^K}\left[a\right]^x\right) \]
\[ \forall S = \bigwedge_{P \in S} \left(a\langle (P)^\uparrow \rangle, a\left[\left(P\right)^K\right]\right) \]
\[ \exists S = \bigvee_{P \in S} \left(e\langle (P)^\uparrow \rangle, e\left[\left(P\right)^K\right]\right) \]

Table 3.5: Second-order type actions for Orthogonality Pairs.

**Theorem 12** (Orthogonality for Actions).

\[ P, Q \in \mathcal{O} \Rightarrow P \land Q \in \mathcal{O} \mathcal{N} \]
\[ P, Q \in \mathcal{O} \Rightarrow P \lor Q \in \mathcal{O} \mathcal{N} \]
\[ P \in \mathcal{O} \Rightarrow \neg P \in \mathcal{O} \mathcal{N} \]
\[ P, Q \in \mathcal{O} \Rightarrow P \rightarrow Q \in \mathcal{O} \mathcal{N} \]
\[ S \subseteq \mathcal{O} \Rightarrow \forall S \in \mathcal{O} \mathcal{N} \]
\[ S \subseteq \mathcal{O} \Rightarrow \exists S \in \mathcal{O} \mathcal{N} \]

**Proof.** The proof is essentially by repeated use of lemma [7] to establish the orthogonality of the given sets using the saturation of the components of elements of \( \mathcal{O} \); that they are made out of constructors/eliminators is obvious.

**3.5.4 Structural Completion**

§3.30. Term/Co-term Closures  Our aim now is to go the opposite way: take the normal terms and co-terms of a type and determine all those that can be of that type. This entails adding to the normal terms the variables and those (co-) abstractions that we deem suitable—that preserve strong normalization. Because abstractions and co-abstractions reduce in a non-deterministic way, we will have to engage in an iterative process which—upon hitting stability—shall yield abstractions and co-abstractions that are orthogonal in no matter which order one reduces them.

Assume, then, that \( K \subseteq \mathcal{K} \) represents the co-terms that we have thus far determined to be in our interpretation of a type. On the term side we must then have three kinds of terms—variables, the terms formed from the relevant constructors (assumed to be in some set \( T \subseteq \mathcal{T} \)), and all the well-behaved abstractions (those that when substituted by
co-terms in $K$ yield cuts that are strongly normalizing). In its full generality, for each set of terms $T$, we define a function

$$[T](K) = Var \cup T \cup \bigcup_{a \in Covar} a.(SN^{|K} \alpha).$$

The result of this operation yields a new set $T'$ which, at best, will only be orthogonal with the given $K$ if we prioritize co-abstractions over abstractions—the restriction only guarantees as much. To get the converse—reductions that normalize when we prioritize abstractions over co-abstractions—the dual construction is needed:

$$[K](T) = Covar \cup K \cup \bigcup_{x \in Var} x.(SN^{|T} x).$$

But then, again, this still does not yield a pair of terms and of co-terms whose abstractions and co-abstractions are mutually orthogonal; this will happen if we have a fix-point of this process of repeatedly finding the compatible abstractions and co-abstractions. The existence of fix-points like these (on the lattice of sets of terms, $P(T)$) is guaranteed by Tarski’s fix-point theorem and the following result.

**Proposition 10** (Contra-variance of Restriction). For any $T \subseteq T$ and $K \subseteq K$ the functions

$$[T] : P(K) \to P(T) \quad \text{and} \quad [K] : P(T) \to P(K)$$

are contra-variant under inclusion of sets. Consequently, the compositions

$$[T] \circ [K] : P(T) \to P(T) \quad \text{and} \quad [K] \circ [T] : P(K) \to P(K)$$

are monotone (covariant) functions.

**Proof.** The contra-variance follows from the contra-variance of the operator associated with the abstractions. Consider any two sets of co-terms $L \subseteq L'$ and a co-variable $\alpha$; then

$$\alpha.(c) \in \alpha.(SN^{|L'} \alpha) \implies \forall l \in L'.c[l/\alpha] \in SN^{|L'}$$

$$\implies \forall l \in L.c[l/\alpha] \in SN^{|L}$$

$$\implies \alpha.(c) \in \alpha.(SN^{|L} \alpha)$$

$$\implies \alpha.(c) \in \bigcup_{a \in Covar} \alpha.(SN^{|L} \alpha)$$

whence

$$\alpha.(SN^{|L'} \alpha) \subseteq \bigcup_{a \in Covar} \alpha.(SN^{|L} \alpha);$$

by the least upper bound property for the union in $P(T)$, we conclude

$$\bigcup_{a \in Covar} \alpha.(SN^{|L'} \alpha) \subseteq \bigcup_{a \in Covar} \alpha.(SN^{|L} \alpha);$$

and, by monotonicity of union for $Var \cup T \cup –$,

$$[T](L') \subseteq [T](L).$$

The result for sets of co-terms follows dually. \qed
Lemma 8. For any set of co-terms \( L \) (resp. of terms \( U \)), if \( T \) (resp. \( K \)) is saturated, then so is \([T](L)\) (resp. \([K](U)\)).

Proof. The set \( SN^a \) is saturated; hence, by theorem \( [11] \), so is \( SN^L_a \) for any co-term set \( L \); and, by lemma \( [5] \), \( \alpha \cdot ( SN^L_a ) \) is, therefore, also saturated. The set of variables \( Var \) is, trivially, saturated, as is \( T \) (by assumption); the union of these sets for any choice of co-variable in the abstraction

\[
[T](L) = Var \cup T \cup \bigcup_{\alpha \in Covar} \alpha \cdot ( SN^L_a )
\]

is also saturated. (Likewise for the co-term closure.) □

§3.31. Structural Orthogonal Completions From the monotonicity of the compositions above (proposition \( [10] \)) we can therefore define the following structural orthogonal completion on pairs of sets of terms \( T \) and co-term \( K \):

\[
(T \bowtie K) = (lfp([T] \circ [K]), [K] (lfp([T] \circ [K])))
\]

The above fix-point is taken on the lattice of subsets of terms, \( \mathcal{P}(SN^T) \). We shall not need any specific characterization of the fix-point other than it being a fix-point; in particular, as observed by Barbanera and Berardi (1996) already, there is no need to prescribe which fix-point to use, but for definiteness we choose the least one. The qualifier “structural” is justified from the structural rules of \( LK \). The parameter sets \( T \) and \( K \) are meant to encode those terms of a type arising from constructors and eliminators; the function then completes this with (co-) variables and (co-) abstractions—those terms whose Curry-Howard equivalents are to be found in the structural (and identity) rules of \( LK \).

Lemma 9. For an arbitrary set of of terms \( T \) and an arbitrary set of co-terms \( K \), let \( P = (T \bowtie K) \). The following equalities characterize the completion:

\[
(P)^T = [T]( (P)^K ) \quad (P)^K = [K]( (P)^T ) .
\]

And from these if follows easily that

\[
Var, T \subseteq (P)^T \quad Covar, K \subseteq (P)^K.
\]

Proof.

\[
(P)^K = [K]( lfp([T] \circ [K]) ) \\
= [K]( (P)^T )
\]

and, then, immediately

\[
= Covar \cup K \cup \bigcup_{x \in Var} x \cdot ( SN^x_a |_{(P)^T} ) \\
\supseteq Covar, K .
\]
The other half of the lemma follows easily from the fix-point property:

\[(P)^T = \text{lfp}([T] \circ [K]) = [T]([K]((P)^T)) = [T]((P)^K)
\]

and, again

\[
\begin{align*}
&= \text{Var} \cup T \cup \bigcup_{a \in \text{Covar}} \alpha \left( S\mathcal{N} \big|_{a}^{(P)^K} \right) \\
&\supseteq \text{Var}, T.
\end{align*}
\]

The following proposition is paramount: it sets, in black and white, that the completion follows the intuitions that guided its definition.

**Proposition 11.** Let \(N \in \mathcal{OPN}\) be an orthogonal normal pair. Its structural completion is an orthogonal pair:

\[\mathbb{N} \triangleq N = ((N)^T) \sqcup ((N)^K) \in \mathcal{OP}\]

**Proof.** Let \(P = \mathbb{N} \triangleq N\); by lemma \(9\), we have that

\[
(P)^T = [(N)^T]((P)^K), \quad \text{and} \quad (P)^K = [(N)^K]((P)^T).
\]

The components of \(N\)—it being an orthogonal pair—are saturated; saturation is then a consequence of lemma \(8\). Consequently, by lemma \(8\), it suffices to prove orthogonality w.r.t. head reductions for

\[
\text{Var} \cup (N)^T \cup \bigcup_{a \in \text{Covar}} \alpha \left( S\mathcal{N} \big|_{a}^{(P)^K} \right) \subseteq S\mathcal{N}T \quad \text{and} \quad \text{Covar} \cup (N)^K \cup \bigcup_{x \in \text{Var}} x \left( S\mathcal{N} \big|_{x}^{(P)^T} \right) \subseteq S\mathcal{N}\mathcal{C}.
\]

This gives a grand total of 9 cases to consider:

Three of them are easy—they lead to cuts that cannot be head reduced:

\[\text{Var} \bullet \text{Covar} \subseteq S\mathcal{N}, \quad \text{Var} \bullet (N)^K \subseteq S\mathcal{N}, \quad \text{and} \quad (N)^T \bullet \text{Covar} \subseteq S\mathcal{N}.
\]

By the assumption on \(N \in \mathcal{OPN}\), we further have

\[(N)^T \bullet (N)^K \subseteq S\mathcal{N}.
\]

For \(t \in (P)^T\) a co-abstraction \(\alpha(c)\), it must be the case that \(t\) is an element of

\[
\bigcup_{a \in \text{Covar}} \alpha \left( S\mathcal{N} \big|_{a}^{(P)^K} \right).
\]

Then, for \(k \in (P)^K\) not an abstraction (i.e. in \(\text{Covar} \cup (N)^K \subseteq (P)^K\)) we have that

\[
\alpha(c) \cdot k \Rightarrow c[k/\alpha] \in S\mathcal{N}
\]
by the restriction. Dually, if \( k \) is an abstraction \( x.(d) \), and \( t \in \text{Var} \cup (N)^{T} \subseteq (P)^{T} \), the restriction operation guarantees that
\[
t \cdot x.(d) \rightsquigarrow d \frac{t}{x} \in SN
\]
Now if both \( t = a.(c) \) and \( k = x.(d) \) are abstractions we have to consider two possible reduction steps. However, by the same reasoning as above, because \( k \) is in \( (P)^{K} \), then \( c[k/a] \in SN \); and—rolling back on our use of the fix-point property on \( t \)— we have \( t \in (P)^{T} \) and so \( d \frac{t}{x} \in SN \)—as we needed.

3.5.5 Interpretation of Types

§3.32. Contexts We now have all the tools we need to build interpretation for types using their constructors. To account for those types with no constructors—i.e. type variables—we need to receive this information for the outside. The interpretation will be parametrized by a context that maps (at least) every free variable in the interpreted type to a given interpretation as an orthogonal pair. These contexts will usually be denoted by letter \( \gamma \). Whenever we wish to add the interpretation of a variable \( X \) to a context we will denote the resulting extension by \( \gamma[X \mapsto \cdot] \).

§3.33. Interpretations Given a type \( T \) and a mapping \( \gamma : \text{ftv}(T) \rightarrow \mathcal{ONP} \) (the context) define the following interpretations by mutual induction on the structure of \( T \):
\[
[T](\gamma) : \mathcal{ONP}
\]
\[
[X](\gamma) = \gamma(X)
\]
\[
[A \land B](\gamma) = \langle A \rangle(\gamma) \land \langle B \rangle(\gamma)
\]
\[
[A \lor B](\gamma) = \langle A \rangle(\gamma) \lor \langle B \rangle(\gamma)
\]
\[
[\neg A](\gamma) = \neg \langle A \rangle(\gamma)
\]
\[
[A \rightarrow B](\gamma) = \langle A \rangle(\gamma) \rightarrow \langle B \rangle(\gamma)
\]
\[
[A \leftarrow B](\gamma) = \langle A \rangle(\gamma) \leftarrow \langle B \rangle(\gamma)
\]
\[
[\forall X. A](\gamma) = \forall \{ \langle A \rangle(\gamma[\gamma(X \mapsto N)]) \mid N \in \mathcal{ONP} \}
\]
\[
[\exists X. A](\gamma) = \exists \{ \langle A \rangle(\gamma[\gamma(X \mapsto N)]) \mid N \in \mathcal{ONP} \}
\]
\[
(\langle T \rangle)(\gamma) : \mathcal{OP}
\]
\[
(\langle T \rangle)(\gamma) = \mathbb{1}(\langle T \rangle(\gamma))
\]

The first one will be the normal interpretation; the second, the completed interpretation.

Theorem 13 (Well-definedness). For any DC type \( T \) and for any suitable interpretation context \( \gamma (\text{ftv}(T) \subseteq \text{dom}(\gamma)) \):
\[
[T](\gamma) \in \mathcal{ONP} \quad \text{and} \quad \langle T \rangle(\gamma) \in \mathcal{OP}
\]

Proof. We can reason by mutual induction on the structure of \( T \) and by prioritizing the interpretation in \( \mathcal{ONP} \) over the one on \( \mathcal{OP} \) since the former requires the latter only for sub-types, and the latter requires the former at the same type. A few cases:
REDUCTION PROPERTIES OF THE DUAL CALCULUS

Type Variables: For a type variable $X$, quite trivially we have
\[ [X](\gamma) = \gamma(X) \in \mathcal{ONP} \]
by the condition on contexts.

Conjunction:
\[
\begin{align*}
\langle A \rangle(\gamma) &\in \mathcal{OP} \text{ and } \langle B \rangle(\gamma) \in \mathcal{OP} & \quad \text{(IH)} \\
\implies \langle A \rangle(\gamma) \land \langle B \rangle(\gamma) &\in \mathcal{ONP} & \quad \text{ (theorem 12)} \\
\implies [A \land B](\gamma) &\in \mathcal{ONP}
\end{align*}
\]

Universal Quantification: For quantifications $T = \forall X . A$ we need to tinker a little with the induction hypothesis. We first need to prove that
\[
S = \{ \langle A \rangle(\gamma[X \mapsto N]) \mid N \in \mathcal{ONP} \} \subseteq \mathcal{OP};
\]
but the free variables of $A$ are (at most) those in $T$ together with $X$; then for $N \in \mathcal{OP}$, \( \langle A \rangle(\gamma[X \mapsto P]) \) falls within the scope of the induction as being an element of $\mathcal{OP}$, and so:
\[
\begin{align*}
S &\subseteq \mathcal{OP} \\
\implies \forall S &\in \mathcal{ONP} & \quad \text{(theorem 12)} \\
\implies [\forall X . A](\gamma) &\in \mathcal{ONP}
\end{align*}
\]

Existential Quantification: On the existential side, the proof goes through the same steps but with meets replaced by joins. So, induction hypothesis says that
\[
S = \{ \langle A \rangle(\gamma[X \mapsto N]) \mid N \in \mathcal{ONP} \} \subseteq \mathcal{OP},
\]
whence,
\[
\begin{align*}
S &\subseteq \mathcal{OP} \\
\implies \exists S &\in \mathcal{ONP} & \quad \text{(theorem 12)} \\
\implies [\exists X . A](\gamma) &\in \mathcal{ONP}
\end{align*}
\]

Complete Interpretation: By the mutual induction hypothesis we get that
\[
[T](\gamma) \in \mathcal{ONP} \implies \downarrow([T](\gamma)) \in \mathcal{OP} \quad \text{(proposition 11)}
\]
\[
\implies \langle T \rangle(\gamma) \in \mathcal{OP}
\]

Lemma 10. The two interpretations can be easily related in the following way:
\[
\begin{align*}
\text{Var}, \ (\downarrow([T](\gamma)))^\top &\subseteq (\langle T \rangle(\gamma))^\top \\
\text{Covar}, \ ([T](\gamma))^K &\subseteq (\langle T \rangle(\gamma))^K
\end{align*}
\]
STRONG NORMALIZATION

Proof. This is just a direct corollary of lemma 9, that the completion includes variables and the sets on which it is parametrized. E.g.,

\[(\langle T \rangle(\nu))^{\top} = \mathbb{1}(\langle [T] \rangle(\nu))\]

\[= \left(\left(\langle [T] \rangle(\nu)\right)^{\top} \parallel \left(\langle [T] \rangle(\nu)\right)^{k}\right)^{\top}\]

\[\supseteq \left([T] \right)(\nu), Var\]

\[\square\]

§3.34. Second-order Properties There are two fairly standard properties that need to be satisfied by every model of a second-order language: type weakening and substitution. The first expresses the invariant that the interpretation depends only on the interpretation of the type variables that appear in a type—and is used to justify type abstraction. Substitution, on the other hand, applies to type instantiations. Interestingly, the proof of substitution requires the weakening property.

Lemma 11 (Weakening). Let \( N \in \mathcal{ONP} \) and \( Y \notin ftv(T) \):

\[[T](\nu[Y \mapsto N]) = [T](\nu)\]

\[\langle T \rangle(\nu[Y \mapsto N]) = \langle T \rangle(\nu)\]

Proof. We prove this by mutual induction on the structure of types, prioritizing the interpretation into \( \mathcal{ONP} \). We analyse some cases below.

Variables: We need only to consider variables \( T = X \neq Y \), for otherwise \( Y \) would be free in \( T \).

\[[X](\nu[Y \mapsto N]) = \nu[Y \mapsto N](X)\]

\[= \nu(X)\]

\[= [X](\nu)\]

Conjunction:

\[[A \land B](\nu[Y \mapsto N]) = \langle A \rangle(\nu[Y \mapsto N]) \land \langle B \rangle(\nu[Y \mapsto N])\]

\[= \langle A \rangle(\nu) \land \langle B \rangle(\nu)\] (IH)

\[= [A \land B](\nu)\]
Universal Quantification: 

$$[\forall X . A](\gamma[Y \mapsto N]) = \forall\{ \langle A \rangle(\gamma[Y \mapsto N][X \mapsto N']) \mid N' \in \mathcal{NP} \}$$

As we identify types up to \(\alpha\)-renaming, we can assume that \(X \neq Y\), so that extensionally 

$$\gamma[Y \mapsto P][X \mapsto P'] = \gamma[X \mapsto P'][Y \mapsto P],$$

leading to

$$= \forall\{ \langle A \rangle(\gamma[X \mapsto N'][Y \mapsto N]) \mid N' \in \mathcal{NP} \} \quad \text{(IH)}$$

Existential Quantification: By the same reasoning as above:

$$[\exists X . A](\gamma[Y \mapsto N]) = \exists\{ \langle A \rangle(\gamma[Y \mapsto N][X \mapsto N']) \mid N' \in \mathcal{NP} \}$$

$$= \exists\{ \langle A \rangle(\gamma[X \mapsto N'][Y \mapsto N]) \mid N' \in \mathcal{NP} \} \quad \text{(IH)}$$

Completed Interpretation:

$$\langle T \rangle(\gamma[Y \mapsto N]) = \llbracket([T]\langle Y \mapsto N \rangle)\rrbracket$$

$$= \llbracket([T]\langle \gamma \rangle)\rrbracket \quad \text{(IH)}$$

$$= \langle T \rangle(\gamma) \quad \square$$

Lemma 12 (Substitution). Let \(T, T'\) be types and \(\text{dom}(\gamma) \subseteq \text{ftv}(T, T') - \{Y\}:\n
$$[T[T'/Y]](\gamma) = [T](\gamma[Y \mapsto [T'](\gamma)])$$

$$\langle T[T'/Y] \rangle (\gamma) = \langle T \rangle(\gamma[Y \mapsto [T'](\gamma)])$$

Proof. Reason by (the usual) mutual induction on the structure of types. As usual, some cases below.

Variables: For \(T = Y\)

$$[Y[T'/Y]](\gamma) = [T'](\gamma)$$

$$= \gamma[Y \mapsto [T'](\gamma)](Y)$$

$$= [Y](\gamma[Y \mapsto [T'](\gamma)]);$$
Subtraction:

$[X[T'/Y]](\gamma) = [X](\gamma)
= \gamma(X)
= \gamma[Y \mapsto [T'](\gamma)](X)
= [X](\gamma[Y \mapsto [T'](\gamma)])$

Existential Quantification: By the same reasoning as above:

$[\exists X . A \, [T'/Y]](\gamma) = [\exists X . A \, [T'/Y]](\gamma)
= \exists \{ A[T'/Y] \, \gamma[X \mapsto N] \mid N \in \mathcal{ONP} \}
= \exists \{ A \, (\gamma[X \mapsto N] \, [Y \mapsto [T'](\gamma)[X \mapsto N])] \mid N \in \mathcal{ONP} \} \quad \text{(IH)}
= \exists \{ A \, (\gamma[X \mapsto N] \, [Y \mapsto [T'](\gamma)]) \mid N \in \mathcal{ONP} \} \quad \text{(weakening)}
= [\exists X . A \, (\gamma[Y \mapsto [T'](\gamma)][X \mapsto N])] \mid N \in \mathcal{ONP}$

Universal Quantification: Using $\alpha$-equivalence for bound type variables in $T$, we posit that, not only are they all different from $Y$, but also that they do not appear free in $T'$. This warrants us pushing substitutions through quantifiers and the use of weakening in the interpretation of $T$.

$[\forall X \, A \, [T'/Y]](\gamma) = [\forall X . A[T'/Y]](\gamma)
= \forall \{ A[T'/Y] \, \gamma[X \mapsto N] \mid N \in \mathcal{ONP} \}
= \forall \{ A \, (\gamma[X \mapsto N] \, [Y \mapsto [T'](\gamma)[X \mapsto N])] \mid N \in \mathcal{ONP} \} \quad \text{(IH)}
= \forall \{ A \, (\gamma[X \mapsto N] \, [Y \mapsto [T'](\gamma)]) \mid N \in \mathcal{ONP} \} \quad \text{(weakening)}
= [\forall X . A \, (\gamma[Y \mapsto [T'](\gamma)])]$
**Completed Interpretation:**

\[
\langle T \!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!\!
\| T' / X \| \downarrow (\gamma) = \sqcup ([T | T' / X]) (\gamma)
\]

\[
= \sqcup ([T | \gamma [X \mapsto [T'] (\gamma)])
\]

\[
= \langle T \| \gamma [X \mapsto [T'] (\gamma)) \rangle
\]

\[
\square
\]

**Lemma 13 (Conservation).** The syntactic introduction and elimination operators of the second-order Dual Calculus conserve the interpretations in the following sense:

\[
\langle (\langle A \| \gamma \rangle)^T, (\langle B \| \gamma \rangle)^K \rangle \subseteq (\langle A \land B \| \gamma \rangle)^T
\]

\[
\text{fst}(\langle A \| \gamma \rangle)^T \subseteq (\langle A \land B \| \gamma \rangle)^K
\]

\[
\text{snd}(\langle A \| \gamma \rangle)^K \subseteq (\langle A \land B \| \gamma \rangle)^K
\]

\[
\text{fst}(\langle A \| \gamma \rangle)^K \subseteq (\langle A \lor B \| \gamma \rangle)^T
\]

\[
\text{snd}(\langle A \| \gamma \rangle)^T \subseteq (\langle A \lor B \| \gamma \rangle)^K
\]

\[
\text{not}(\langle A \| \gamma \rangle)^T \subseteq (\langle \lnot A \| \gamma \rangle)^T
\]

\[
\text{not}(\langle A \| \gamma \rangle)^K \subseteq (\langle \lnot A \| \gamma \rangle)^K
\]

\[
\lambda x. (\langle B \| \gamma \rangle)^K_x^{\langle A \| \gamma \rangle (\gamma)} \subseteq (\langle A \rightarrow B \| \gamma \rangle)^T
\]

\[
(\langle A \| \gamma \rangle)^T \# (\langle B \| \gamma \rangle)^K \subseteq (\langle A - B \| \gamma \rangle)^T
\]

\[
\mu a. (\langle A \| \gamma \rangle)^K_{\langle B \| \gamma \rangle (\gamma)} \subseteq (\langle A - B \| \gamma \rangle)^K
\]

for \( S = \{ \langle A \| \gamma [X \mapsto N] \mid N \in \mathcal{ONP} \} \):

\[
a \left( \bigcap_{P \in S} (P)^T \right) \subseteq (\forall X . A \| \gamma \rangle)^T
\]

\[
a \left( \bigcup_{P \in S} (P)^K \right) \subseteq (\forall X . A \| \gamma \rangle)^K
\]

\[
e \left( \bigcap_{P \in S} (P)^T \right) \subseteq (\exists X . A \| \gamma \rangle)^T
\]

\[
e \left( \bigcup_{P \in S} (P)^K \right) \subseteq (\exists X . A \| \gamma \rangle)^K
\]

**Proof.** The proof of all of these are just applications of the definitions of the (normal) interpretation and of the type actions together with lemma 10. E.g.,

\[
e \left( \bigcup_{P \in S} (P)^T \right) \subseteq \bigcup_{P \in S} e \langle (P)^T \rangle
\]

\[
= (\exists \{ \langle A \| \gamma [X \mapsto N] \mid N \in \mathcal{ONP} \})^T
\]

\[
= (\exists X . A \| \gamma \rangle)^T
\]

\[
\subseteq (\exists X . A \| \gamma \rangle)^K
\]
The (co-) abstraction syntactic operations orthogonalize—with respect to strongly normalizing cuts—interpretations in the sense that they take the interpretation for terms in order to construct a subset of the interpretation for co-terms.

\[ \alpha \left( \mathcal{SN}^{\{\big\}_{\alpha}(\gamma)^{\big\}} \right) \subseteq (\mathcal{A}(\gamma))^T \quad \text{and} \quad \pi \left( \mathcal{SN}^{\{\big\}_{\pi}(\gamma)^{\big\}} \right) \subseteq (\mathcal{A}(\gamma))^K \]

**Proof.**

\[ \alpha \left( \mathcal{SN}^{\{\big\}_{\alpha}(\gamma)^{\big\}} \right) \subseteq \bigcup_{\alpha \in \text{Covar}} \alpha \left( \mathcal{SN}^{\{\big\}_{\alpha}(\gamma)^{\big\}} \right) \]

\[ \subseteq \left[ (\mathcal{A}(\gamma))^T \right] (\mathcal{A}(\gamma))^K \]

\[ \subseteq (\mathcal{A}(\gamma))^T \]

\[ x \left( \mathcal{SN}^{\{\big\}_{x}(\gamma)^{\big\}} \right) \subseteq \bigcup_{x \in \text{Var}} x \left( \mathcal{SN}^{\{\big\}_{x}(\gamma)^{\big\}} \right) \]

\[ \subseteq \left[ (\mathcal{A}(\gamma))^K \right] (\mathcal{A}(\gamma))^T \]

\[ \subseteq (\mathcal{A}(\gamma))^K \]

\[ = (\mathcal{A}(\gamma))^K \]

### 3.5.6 Strong Normalization

§3.35. **General Working** The issue with trying to prove that a calculus such as DC is strongly normalizing by a direct induction is that, via substitution, the reduction of abstractions leads to phrases that are not syntactically smaller and to which we are not entitled to use our inductive knowledge of strong normalization. Variables are central to this issue: they stand for arbitrary (if well-typed) terms, and these terms themselves
can have other variables that may, at later stages, be substituted by other such terms, ad infinitum. A similar reasoning applies to co-variables. Realizability corresponds to a generalization of the induction hypothesis where we assume that not only is the phrase under consideration strongly normalizing, but it remains strongly normalizing for any substitution that contains its free (co-)variables. The phrases being substituted in need, themselves, to obey the same invariant; otherwise, it would be quite easy for misbehaved terms to sneak inside our model. In the ensuing, we will define formally the substitutions that are suitable for our intents, and then prove an adequacy result: any well-typed phrase of the Dual Calculus is in our interpretation of types as orthogonal pairs, when under a suitable substitution. Fortunately, the identity substitution is suitable, and so we can further conclude that any well-typed phrase of DC is strongly normalizing.

§3.36. Substitution   A (bi-) substitution is a mapping of variables and co-variables into terms and co-terms in the adequate interpretation; here adequate means in the interpretation of the type they have in the typing context; and, because the interpretation needs to be contextualized, we need also to take the context into account. More formally, for typing contexts $\Gamma$ and co-contexts $\Delta$, and an interpretation $\gamma$ containing all the free type variables of $\Gamma$ and $\Delta$, a substitution for them is a partial mapping whose domain contains $\text{dom}(\Gamma) \cup \text{dom}(\Delta)$ and which satisfies

$$x : T \in \Gamma \implies x \in (\llbracket T \rrbracket(\gamma))^T,$$
$$\alpha : T \in \Delta \implies \alpha \in (\llbracket T \rrbracket(\gamma))^K.$$

The application of a substitution $\sigma$ to a phrase $p$ is denoted by $p[\sigma]$.

**Theorem 14 (Adequacy).** Let $t$, $k$ and $c$ stand for terms, co-terms and cuts of the dual calculus. For any typing contexts and co-contexts $\Gamma$ and $\Delta$, s.t.

$$\Gamma \vdash t : T \mid \Delta \quad \Gamma \vdash k : T \dashv \Delta \quad \Gamma \vdash c \dashv \Delta;$$

for any (suitable) interpretation context $\gamma$ for $\Gamma$, $\Delta$ and $A$, and correspondingly suitable substitution $\sigma$, we have that

$$t[\sigma] \in (\llbracket T \rrbracket(\gamma))^T, \quad k[\sigma] \in (\llbracket T \rrbracket(\gamma))^K \quad \text{and} \quad c[\sigma] \in SN.$$

**Proof.** By rule induction on the typing trees.

**(Co-) Variables:**

$$x[\sigma] = \sigma(x) \in (\llbracket T \rrbracket(\gamma))^T$$
$$\alpha[\sigma] = \sigma(\alpha) \in (\llbracket T \rrbracket(\gamma))^K$$

**(Co-) Abstractions:** Term abstractions $\Gamma \mid x.(c) : A \dashv \Delta$ lead to a complication with the handling of the induction hypothesis. We assume we are given a substitution $\sigma$ for $\Gamma$, $\Delta$ and $\gamma$. We cannot simply apply it to $c$, for the typing context for $c$ contains $x : A$ (which, by $\alpha$-renaming, we assume is not in the domain of $\sigma$). However, for any $t \in (\llbracket A \rrbracket(\gamma))^T$, the extended substitution $\sigma[t/x]$ is in the conditions of the adequacy theorem and we can reason

$$c[\sigma[t/x]] = c[\sigma][t/x] \in SN.$$
STRONG NORMALIZATION

The cut $c[\sigma]$ then satisfies

$$c[\sigma] \in SN \bigg|_{x}^{(\langle A \rangle(\gamma))^T}$$

whence we conclude

$$\Rightarrow x.(c[\sigma]) \in x.\left(SN \bigg|_{x}^{(\langle A \rangle(\gamma))^T}\right)$$

$$\Rightarrow x.(c[\sigma]) \in (\langle A \rangle(\gamma))^K$$

$$\Rightarrow x.(c)[\sigma] \in (\langle A \rangle(\gamma))^K$$

Co-abstractions follow similarly. This time, for any co-term $k \in (\langle A \rangle(\gamma))^K$ the induction hypothesis yields

$$c[\sigma[k/a]] = c[\sigma][k/a] \in SN;$$

whence,

$$c[\sigma] \in SN \bigg|_{x}^{(\langle A \rangle(\gamma))^K}$$

$$\Rightarrow \alpha.(c[\sigma]) \in \alpha.\left(SN \bigg|_{x}^{(\langle A \rangle(\gamma))^K}\right)$$

$$\Rightarrow \alpha.(c[\sigma]) \in (\langle A \rangle(\gamma))^T$$

$$\Rightarrow \alpha.(c)[\sigma] \in (\langle A \rangle(\gamma))^T$$

**Thinning:** For left thinning, we are given a substitution $\sigma$ for some $x : A, \Gamma$ and $\Delta$ and $\gamma$; as $\Gamma$ has at most as many free type variables as $x : A, \Gamma$, $\gamma$ is also a suitable interpretation context for the latter. Substitution $\sigma$ is then a substitution for $\Gamma$ as

$$(y : B) \in \text{dom}(\Gamma) \Rightarrow (y : B) \in \text{dom}(x : A, \Gamma) \Rightarrow \sigma(y) \in (\langle B \rangle(\gamma))^T;$$

whence, by their respective induction hypothesis, any $t$, $k$, and $c$

$$t[\sigma] \in (\langle T \rangle(\gamma))^T, \quad k[\sigma] \in (\langle T \rangle(\gamma))^K \quad \text{and} \quad c[\sigma] \in SN,$$

which, as the phrases do not change in the statement of the weakening rules, are exactly what we need.

Similarly, any interpretation context for $\Gamma$ and $\Delta, \alpha : A$ is also suitable for $\Delta$; and any substitution suitable for these contexts is suitable for $\Gamma$, $\Delta$ and $\gamma$. Whence, the induction hypothesis is enough to establish our intended inclusions (the same as above).

**Interchange:** The set of (co-) variables in the contexts of an interchange rule do not change—only their order does. This is irrelevant for our definition of suitability. Likewise, the set of free type variables is invariant during exchange. As in the case of thinning, then, any substitution suitable for the consequent of a rule is also suitable for the antecedent and, because the phrase of each type of judgement does not change from consequent to antecedent, all we need to do is apply the induction hypothesis.
Contraction: In (right) thinning, we start with a big context and re-use it with the smaller one, contraction offers the opposite challenge: we start with a small context \( x : A, \Gamma \) and must—if we are to use the induction hypothesis—adapt it to one with more elements, \( x : A, y : A, \Gamma \). That being said, the interpretation context \( \gamma \) that we start with can remain the same as the set of free type variables is invariant:

\[
\operatorname{dom}(\gamma) \subseteq \text{ftv}(x : A, \Gamma) = \text{ftv}(A) \cup \text{ftv}(\Gamma) = \text{ftv}(A) \cup \text{ftv}(A) \cup \text{ftv}(\Gamma) = \text{ftv}(x : A, y : B, \Gamma).
\]

Now, because the left typing context has an extra variable, \( y \), that we are not guaranteed to have in \( \operatorname{dom}(\sigma) \), we need to extend it with something in \( \{A\} \). \( \gamma \)---luckily, because \( x : A \), we have that \( \sigma(x) \in \{A\} \). Applying then, the induction hypothesis to the extended substitution \( \sigma[y \mapsto \sigma(x)] \) we get that

\[
t[\sigma[y \mapsto \sigma(x)]] \in (\{T\}(\gamma))^T, \quad k[\sigma[y \mapsto \sigma(x)]] \in (\{T\}(\gamma))^K, \quad \text{and} \quad c[\sigma[y \mapsto \sigma(x)]] \in SN.
\]

From this we conclude

\[
t[x/y][\sigma] \in (\{T\}(\gamma))^T, \quad k[x/y][\sigma] \in (\{T\}(\gamma))^K, \quad \text{and} \quad c[x/y][\sigma] \in SN.
\]

Right contraction is proven similarly.

Conjunction: For the term side of conjunction, the antecedents of the typing rule are \( \Gamma \vdash t : A | \Delta \) and \( \Gamma \vdash t' : B | \Delta \); the conclusion is \( \Gamma \vdash \langle t, t' \rangle : A \land B | \Delta \). Any interpretation context \( \gamma \) and substitution \( \sigma \) that satisfy the pre-conditions of the theorem for the conclusion also satisfy them for the antecedents. From the induction hypothesis then, we can reason:

\[
\begin{align*}
t[\sigma] &\in (\{A\}(\gamma))^T \quad \text{and} \quad t'[\sigma] \in (\{B\}(\gamma))^T \quad \text{(IH)} \\
\implies \langle t[\sigma], t'[\sigma] \rangle &\in (\{A \land B\}(\gamma))^T \\
\implies \langle t, t' \rangle[\sigma] &\in (\{A \land B\}(\gamma))^T
\end{align*}
\]

\[
\begin{align*}
k[\sigma] &\in (\{A\}(\gamma))^K \quad \text{(IH)} \\
\implies \text{fst}[k[\sigma]] &\in (\{A \land B\}(\gamma))^K \\
\implies \text{fst}[k][\sigma] &\in (\{A \land B\}(\gamma))^K
\end{align*}
\]

\[
\begin{align*}
k[\sigma] &\in (\{B\}(\gamma))^K \quad \text{(IH)} \\
\implies \text{snd}[k[\sigma]] &\in (\{A \land B\}(\gamma))^K \\
\implies \text{snd}[k][\sigma] &\in (\{A \land B\}(\gamma))^K
\end{align*}
\]
STRONG NORMALIZATION

Disjunction:

\[ t[\sigma] \in (\langle A \rangle(\gamma))^T \]  
\[ \implies t_1\langle t[\sigma] \rangle \in (\langle A \lor B \rangle(\gamma))^T \]  
\[ \implies t_1\langle t[\sigma] \rangle \in (\langle A \lor B \rangle(\gamma))^T \]

\[ t[\sigma] \in (\langle B \rangle(\gamma))^T \]  
\[ \implies t_2\langle t[\sigma] \rangle \in (\langle A \lor B \rangle(\gamma))^T \]  
\[ \implies t_2\langle t[\sigma] \rangle \in (\langle A \lor B \rangle(\gamma))^T \]

\[ k[\sigma] \in (\langle A \rangle(\gamma))^K \quad \text{and} \quad k'[\sigma] \in (\langle B \rangle(\gamma))^K \]  
\[ \implies [k[\sigma], k'[\sigma]] \in (\langle A \lor B \rangle(\gamma))^K \]  
\[ \implies [k, k'][\sigma] \in (\langle A \lor B \rangle(\gamma))^K \]

Negation:

\[ k[\sigma] \in (\langle A \rangle(\gamma))^K \]  
\[ \implies \text{not}(k[\sigma]) \in (\langle \neg A \rangle(\gamma))^T \]  
\[ \implies \text{not}(k[\sigma]) \in (\langle \neg A \rangle(\gamma))^T \]

\[ t[\sigma] \in (\langle A \rangle(\gamma))^T \]  
\[ \implies \text{not}[t[\sigma]] \in (\langle \neg A \rangle(\gamma))^K \]  
\[ \implies \text{not}[t[\sigma]] \in (\langle \neg A \rangle(\gamma))^K \]

Implication: By virtue of \( \alpha \)-renaming, we assume \( x \) does not appear free in the image of \( \sigma \). We can, thus, separate the mutual substitution into \( \sigma \) and \( x \) which allow us to apply the induction hypothesis for any choice of \( t' \in (\langle A \rangle(\gamma))^T \) to get

\[ t[\sigma[t'/x]] = t[\sigma][t'/x] \in (\langle B \rangle(\gamma))^T; \]

we conclude

\[ t[\sigma] \in (\langle B \rangle(\gamma))^T \bigcup^{\langle A \rangle(\gamma)}; \]

whence by conservation

\[ (\lambda x. t[\sigma]) \in \lambda x. (\langle B \rangle(\gamma)) \bigcup^{\langle A \rangle(\gamma)}; \]
REDUCTION PROPERTIES OF THE DUAL CALCULUS

\[ (\lambda x.(t[\sigma])) \in (\langle A \rightarrow B \rangle(\gamma))^T \]
\[ (\lambda x.(t))[\sigma] \in (\langle A \rightarrow B \rangle(\gamma))^T \]

\[ t[\sigma] \in (\langle A \rangle(\gamma))^T \quad \text{and} \quad k[\sigma] \in (\langle B \rangle(\gamma))^K \]
\[ \Rightarrow (t[\sigma]@k[\sigma]) \in (\langle A \rightarrow B \rangle(\gamma))^K \]
\[ \Rightarrow ((t@k))[\sigma] \in (\langle A \rightarrow B \rangle(\gamma))^K \]

**Subtraction:**

\[ t[\sigma] \in (\langle A \rangle(\gamma))^T \quad \text{and} \quad k[\sigma] \in (\langle B \rangle(\gamma))^K \]
\[ \Rightarrow (t[\sigma]\#k[\sigma]) \in (\langle A - B \rangle(\gamma))^T \]
\[ \Rightarrow ((t\#k))[\sigma] \in (\langle A - B \rangle(\gamma))^T \]

On the continuation side, we assume again, co-variable \( \alpha \) is chosen fresh everywhere. As the typing context on the assumption of the typing rule has an extra \( \alpha : B \), we have that for any substitution \( \sigma \) for the conclusion and \( k' \in (\langle B \rangle(\gamma))^K \), the substitution \( \sigma[k'/\alpha] \) is in the conditions of the theorem, and, therefore the induction hypothesis yields:

\[ k[\sigma[k'/\alpha]] = k[\sigma][k'/\alpha] \in (\langle A \rangle(\gamma))^K; \]

by definition of restriction we conclude

\[ k[\sigma] \in (\langle A \rangle(\gamma))^{K}_{\sigma(\langle B \rangle(\gamma))} \; ; \]

and thereby

\[ \mu \alpha.(k[\sigma]) \in \mu \alpha.((\langle A \rangle(\gamma))^{K}_{\sigma(\langle B \rangle(\gamma))}) \]
\[ \Rightarrow \mu \alpha.(k[\sigma]) \in (\langle A - B \rangle(\gamma))^K \]
\[ \Rightarrow \mu \alpha.(k)[\sigma] \in (\langle A - B \rangle(\gamma))^K \]

**Universal Quantification:** For universal quantification we need to be careful in the application of the inductive hypothesis. If \( a(t) \) has type \( T = \forall X . A \) then it must be that \( \Gamma \vdash t : A \mid \Delta \), and that \( X \) is not free in neither \( \Gamma \), nor \( \Delta \); however, \( X \) may appear free in \( A \), and hence the interpretation of \( A \) must be contextualized with a value for this type variable. The image of substitution \( \sigma \) has phrases interpreted only under context \( \gamma \), which does not contain \( X \) (as we assume it is chosen fresh)—and, so, a priori, the interpretation may be ill-defined. However, because \( X \) does not appear free in \( \Gamma, \Delta \), by lemma \([11]\), we know that no matter which \( N \in \mathcal{ONP} \) we use to interpret \( X \), the result is always the same; hence

\[ x \in \text{dom}(\Gamma) \Rightarrow \sigma(x) \in (\langle \Gamma(x) \rangle(\gamma))^T \quad \text{iff} \quad x \in \text{dom}(\Gamma) \Rightarrow \sigma(x) \in (\langle \Gamma(x) \rangle(\gamma[X \mapsto N]))^T \]
\[ \alpha \in \text{dom}(\Delta) \implies \sigma(\alpha) \in (\langle \Delta(\alpha) \rangle(\gamma))^K \text{ iff } \alpha \in \text{dom}(\Delta) \implies \sigma(x) \in (\langle \Delta(x) \rangle(\gamma[X \mapsto N]))^K \]

Substitution \( \sigma \) is, therefore, in the conditions of the theorem for any extension \( \gamma[X \mapsto N] \), and we can use the induction hypothesis to reason

for all \( N \in \mathcal{ONP} \), \( t[\sigma] \in (\langle A \rangle(\gamma[X \mapsto N]))^T \)

\( \implies t[\sigma] \in \bigcap_{P \in \{ \langle A \rangle(\gamma[X \mapsto N]) \} \mid N \in \mathcal{ONP}} \bigcup_{P \in \{ \langle A \rangle(\gamma[X \mapsto N]) \} \mid N \in \mathcal{ONP}} (P)^T \)

\( \implies a(t[\sigma]) \in a\left( \bigcap_{P \in \{ \langle A \rangle(\gamma[X \mapsto N]) \} \mid N \in \mathcal{ONP}} \bigcup_{P \in \{ \langle A \rangle(\gamma[X \mapsto N]) \} \mid N \in \mathcal{ONP}} (P)^T \right) \)

The co-term side does not require such a complicated argument; it does require the use of the other lemma, the substitution one (lemma 12)

\[ k[\sigma] \in (\langle A[T/X] \rangle(\gamma))^K \]

\( \implies k[\sigma] \in (\langle A \rangle(\gamma[X \mapsto [T](\gamma)]))^K \)

\( \implies k[\sigma] \in \bigcup_{P \in \{ \langle A \rangle(\gamma[X \mapsto N]) \} \mid N \in \mathcal{ONP}} (P)^K \)

\( \implies a[k[\sigma]] \in a\left( \bigcup_{P \in \{ \langle A \rangle(\gamma[X \mapsto N]) \} \mid N \in \mathcal{ONP}} (P)^K \right) \)

\[ a[k[\sigma]] \in (\langle \forall X . A \rangle(\gamma))^K \]

\[ a[k][\sigma] \in (\langle \forall X . A \rangle(\gamma))^K \]

**Existential Quantification:** Dually, the proof for the existential quantification case requires the substitution lemma for the term side, and the co-term side requires the same argument as above to justify the application of the induction hypothesis.

\[ t[\sigma] \in (\langle A[T/X] \rangle(\gamma))^K \]

\( \implies t[\sigma] \in (\langle A \rangle(\gamma[X \mapsto [T](\gamma)]))^K \)

\( \implies t[\sigma] \in \bigcup_{P \in \{ \langle A \rangle(\gamma[X \mapsto N]) \} \mid N \in \mathcal{ONP}} (P)^T \)

\( \implies e(t[\sigma]) \in e\left( \bigcup_{P \in \{ \langle A \rangle(\gamma[X \mapsto N]) \} \mid N \in \mathcal{ONP}} (P)^T \right) \)
REDUCTION PROPERTIES OF THE DUAL CALCULUS

\[ \Rightarrow e(t[\sigma]) \in (\exists X . A)(\gamma)^{T} \]
\[ \Rightarrow e(t)[\sigma] \in (\exists X . A)(\gamma)^{T} \]

for all \( N \in \mathcal{ON}P \), \( k[\sigma] \in (\langle A \rangle(\gamma[X \mapsto N]))^{K} \) (IH)
\[ \Rightarrow k[\sigma] \in \bigcap_{P \in \{ \langle A \rangle(\gamma[X \mapsto N]) \mid N \in \mathcal{ON}P \}} (P)^{K} \]
\[ \Rightarrow e[k[\sigma]] \in e\left( \bigcap_{P \in \{ \langle A \rangle(\gamma[X \mapsto N]) \mid N \in \mathcal{ON}P \}} (P)^{K} \right) \]
\[ \Rightarrow e[k[\sigma]] \in (\exists X . A)(\gamma)^{K} \]
\[ \Rightarrow e[k][\sigma] \in (\exists X . A)(\gamma)^{K} \]

Cut: The derivation for cuts brings a difficulty of its very own. Because of the way judgements are represented, the type \( A \) on which the cut is made—i.e. the type of the term and of the co-term of the cut—is not present in the conclusion. It is not necessary, then, that the type variables in \( A \) all present in the typing contexts—and therefore not be covered by the interpretation context \( \gamma \) for the conclusion.

It is not a problem, though, to extend \( \gamma \) exactly with those variables that are in \( A \) but not in \( \text{dom}(\gamma) \) for an arbitrary element of \( \mathcal{ON}P \)—say, \( \perp = (\emptyset, SN\mathcal{K}) \). Explicitly, let \( C = \text{ftv}(A) - \text{dom}(\gamma) \) (a finite set) and represent the extension by
\[ \gamma' = \gamma[C \mapsto \perp]. \]

Any substitution \( \sigma \) in the conditions of the theorem for the conclusion, will satisfy
\[ x : T \in \Gamma \quad \Rightarrow \quad \text{ftv}(T) \subseteq \text{dom}(\gamma) \]
and
\[ a : T' \in \Delta \quad \Rightarrow \quad \text{ftv}(T') \subseteq \text{dom}(\gamma); \]
whence by (a straightforward generalization of) weakening:
\[ \langle T \rangle(\gamma) = \langle T \rangle(\gamma') \quad \text{and} \quad \langle T' \rangle(\gamma) = \langle T' \rangle(\gamma'), \]
and
\[ x : T \in \Gamma \quad \Rightarrow \quad \sigma(x) \in (\langle T \rangle(\gamma))^{T} = (\langle T \rangle(\gamma'))^{T} \]
\[ a : T \in \Delta \quad \Rightarrow \quad \sigma(x) \in (\langle T \rangle(\gamma))^{K} = (\langle T \rangle(\gamma'))^{K} \]

\(^{5}\text{E.g.} \quad x : X \vdash t_{1}(x \cdot y)(x \cdot a) \vdash a : X\)

where the intermediate type is \( X \lor Y \).
STRONG NORMALIZATION

We can therefore apply the induction hypothesis to \( \gamma' \) and \( \sigma \) for \( t \) and \( k \) to yield

\[
\begin{align*}
     t[\sigma] \in \langle A \langle \gamma' \rangle \rangle^T & \quad \text{and} \quad k[\sigma] \in \langle A \langle \gamma' \rangle \rangle^K \\
\implies \quad t[\sigma] \cdot k[\sigma] \in SN \quad \text{(IH)} \\
\implies \quad (t \cdot k)[\sigma] \in SN
\end{align*}
\]

\[\square\]

Corollary 3 (Strong Normalization). Every well-typed phrase of DC is strongly normalizing.

Proof. For any derivable judgement \( \Gamma \vdash c \vdash \Delta \), let \( \gamma \) be the interpretation context mapping \( \text{ftv}(\Gamma, \Delta) \) to \( \bot \)—it satisfies the precondition for adequacy. Define the (obviously identity) substitution \( \text{id} \):

\[
\begin{align*}
     x : T \in \Gamma \implies \text{id}(x) = x \in \text{Var} \subseteq \langle A \langle \gamma \rangle \rangle^T \\
     \alpha : T \in \Delta \implies \text{id}(\alpha) = \alpha \in \text{Covar} \subseteq \langle A \langle \gamma \rangle \rangle^K
\end{align*}
\]

also in the conditions of adequacy. It follows

\[c[\text{id}] = c \in SN.\]

Likewise, any well-typed term \( \Gamma \vdash t : A \mid \Delta \) and any well-typed co-term \( \Gamma \mid k : A \rightarrow \Delta \) are strongly normalizing:

\[t[\text{id}] = t \in \langle A \langle \gamma \rangle \rangle^T \subseteq SN^T \quad \text{and} \quad k[\text{id}] = k \in \langle A \langle \gamma \rangle \rangle^K \subseteq SN^K\]

\[\square\]
REDUCTION PROPERTIES OF THE DUAL CALCULUS
Chapter 4

Mendler Induction

§4.1. Introduction

Before we explain how to extend the Dual Calculus with induction \textit{à la} Mendler \cite{Mendler1991}, it will be worthwhile to focus first on Mendler induction on its own. This particular way of doing induction seems to not have taken a hold on the community. This is in no small part, because it is more elaborate—formally and conceptually—than the usual notion of induction. Also, Mendler induction can be encoded using second-order constructs \cite{UustaluVene1999}.

But Mendler induction is still, we argue, a very interesting research idea: its second-order encoding introduces a lot of complexity (cf. table 4.3); its status as a logic construct is still somewhat hazy; and, from a Computer Science perspective, by giving back the power to determine \textit{when} induction is called, Mendler induction potentially offers two potential benefits: in speed, and in convenience.

This final point can be recast on a more philosophical level. There is a unifying principle in the design of both DC and Mendler induction. On the former, Wadler \cite{Wadler2003} pp. 200) remarked that

\begin{quote}
like the Pompidou Center in Paris, the plumbing is exposed on the outside. While this can make the expression harder on the eyes, it also—like CPS, and like the Pompidou Center—has the advantage of revealing structure that previously was hidden.
\end{quote}

Similarly, Mendler induction operates by handing direct control over the inductive call to the program—whilst avoiding potential ill-effects with the aid of the type system. The plumbing is revealed—and with it, potentially, new, research stimulating, structure.

§4.2. Structure of the Chapter

We will start by studying Mendler induction not as given by Mendler originally, but will instead focus on a syntactic formulation of the categorical presentation of Uustalu and Vene \cite{UustaluVene1999}, as it is—we think—simpler to understand, and easier to relate to ordinary induction. It does however come at the price of having to include existential types in the base language—but these, if not common, are not hard to understand. We will then show how Mendler and ordinary induction can encode one another.

\footnote{Unless otherwise noted, our exposition and results are independent of those of Matthes \cite{Matthes1999} whose work came to the author’s attention late in his Ph.D.—and which also draws the connection with existential types.}
4.1 Mendler’s Induction Operator

§4.3. Base Calculus: System F  The concept of Mendler induction arose from a syntactic study of induction within the scope of System F (Girard, 1971; Girard et al., 1989). For good measure, we review the system in table 4.1. The definition is for the most part standard but there are a few caveats. We include product types as they are important for Mendler induction. Notation-wise, we will often omit the type annotations in the projections. For the purposes of this introduction, we eschew reduction and use, instead, an equational theory based on $\beta$ and $\eta$ conversion.

In the sequel, we will assume the type $1 = \forall X . X \to X$ and its inhabitant

$$* \equiv \lambda X. \lambda x : X.x;$$

and, for a few examples, we will also take the existence of some sort of co-product, for which we can take the usual Church encoding (Girard et al., 1989, 11.3.4). Whenever needed, we will represent injections by the usual $i_t(t)$; the case (or either) operator is to be denoted by $[f; g]$, where $f$ and $g$ are functions from each of the disjuncts with common co-domain.

§4.4. Taming Recursion  The problem with unguarded recursion is that we can apply the recursive call to arbitrary elements—including elements which are “bigger” than those we started with as in

$$f \ n = f \ (n + 1).$$

Mendler noticed that the more expressive type system of System F was capable of restricting the application of functions on a type only to some (pre-screened) terms of that type—and that this could be used to pass recursive or inductive functions around whilst maintaining the safety of the language. This modus operandi is akin to doing data abstraction in programming languages—hiding data from anything other than the public methods of an object or functions defined in the signature of an abstract data type. In System F a solution to this problem is afforded to us by existential types. Another solution is provided by universally quantified types—indeed, this was the route taken by Mendler (1991)—but the existential types version of Mendler induction (Uustalu and Vene, 1999) feels more natural because it separates the two pieces of structure—encapsulation and induction (see section 4.3).

2 Computer Scientists often use the term “recursion operator” for the kind of construction we introduce here. Mendler (1991), however, uses the term inductor (“The constant $R$ is the induction combinator…”; note, though, that the operator is represented by the letter $R$…). This is also consistent with the usage of Girard et al. (1989): recursive steps have access to the value on which the recursive call is made, whereas inductive steps only have access to the result of the inductive call. Recursive functions seem to be so called because they have the same arguments as primitive recursive functions; inductive functions are those which, like simple proofs by induction, only look at the what happens at its immediate predecessors. This neat separation of the two concepts lead us to prefer this terminology. That being said, the confusion about when to use the two terms is widespread—to quote none other than Kleene (1967, p. 209), “Axioms 18-19 and 20-21 [sic] provide ‘recursive definitions’ or ‘definitions by induction of $+$ […] and $\cdot$ […]’.”

3 To borrow the nomenclature of Barendregt (1984).

4 According to Pierce (2002), the connection between abstract data types and existential types was first noted by Mitchell and Plotkin (1988). The same Mitchell was acknowledged by Mendler (1991) in his paper, but it seems that he did not notice the connection as he makes no mention of Mitchell and Plotkin’s work.
Types:

\[ T ::= X \mid T \land T' \mid T \to T' \mid \forall X . T \]

Terms:

\[ t ::= x, y, \ldots \mid \langle t_1, t_2 \rangle \mid \pi_i \mid \lambda x : T . t \mid t_1 \land t_2 \mid \forall X . t \mid t_1 \]  

(i \in 1, 2)

Typing Rules:

\[ x : T \vdash x : T \]

\[ \Gamma \vdash t : A \quad \Gamma \vdash t' : B \quad \frac{}{\Gamma \vdash \langle t, t' \rangle : A \land B} \]

\[ \vdash \pi_i : A_1 \land A_2 \to A_i \]

\[ \Gamma \vdash t : A, \Gamma \vdash t : B \quad \frac{}{\Gamma \vdash \lambda x : A . t : A \to B} \]

\[ \Gamma \vdash t : A \to B \quad \Gamma \vdash t' : A \quad \frac{}{\Gamma \vdash t' : B} \]

\[ \Gamma \vdash t : T \quad (X \not\in \Gamma) \]

\[ \frac{}{\Gamma \vdash \Lambda X : T : t} \]

\[ \Gamma \vdash t : T \quad \frac{}{\Gamma \vdash t T : T'[T'/X]} \]

Conversion:

\[ (\Lambda X . t) T = t[T/X] \]

\[ \pi_i \langle t_1, t_2 \rangle = t_i \]

\[ \lambda x : T . t x =_\eta t \]

\[ \Lambda X . t X =_\eta t \]

\[ (\lambda x : T . t) t' = t[t'/x] \]

\[ \langle \pi_i t_1, \pi_i t_2 \rangle =_\eta t \]

\[ (x \not\in \text{fv}(t)) \]

\[ (X \not\in \text{ftv}(t)) \]

Table 4.1: System F
MENDLER INDUCTION

Types: \( T = \ldots \mid \exists X. T \)

Terms: \( t = \ldots \mid ET.t \mid \text{let } X, x = t \text{ in } t' \)

Typing Rules:

\[
\Gamma \vdash t : F(T) \\
\Gamma \vdash \exists X. F(X) \\
\Gamma, x : F(X) \vdash t' : T \quad (X \text{ and } x \text{ not free in } \Gamma, T)
\]

Conversion:

\[
\begin{align*}
\text{let } X, x = ET.t \text{ in } t' &= t'[T/X][t/x]. \\
\text{let } X, x = t \text{ in } t'(EX.x) &= t't \\
\end{align*}
\]

Table 4.2: Extension of System F with existential types

§4.5. Existential Types in System F  The intuition behind existential types (table 4.2) is not particularly puzzling: a term \( t \) has an existential type \( \exists X. F(X) \) if it can be typed with an instance \( F(T) \) of \( F(X) \)—any and whatever it may be. Formally, to keep the type system syntax directed, we should include an existential quantification operator on terms \( ET.t \) (capital epsilon) together with the following typing rule:

\[
\Gamma \vdash t : \exists X. F(X) \\
\Gamma \vdash ET.t : \exists X. F(X)
\]

The puzzling bit is how to use—i.e., continue from—existential values in general.\(^5\) This we do only if the knowledge of the abstracted type is not used in the continuation. Formally, we have

\[
\Gamma \vdash \exists X. F(X) \quad \Gamma, x : F(X) \vdash t' : T \\
\Gamma \vdash \text{let } X, x = t \text{ in } t' : T,
\]

with the restriction that \( x \) and \( X \) do not appear free in \( \Gamma \) or \( T \).

With this in place, we can define the existential (\( \beta \)-) conversion rule

\[
\text{let } X, x = ET.t \text{ in } t' = t'[T/X][t/x].
\]

Crucially, because \( t' \) can only be well typed if \( x \) is used only with the information that its type \( F \) is instantiated with some unknown type \( X \), it cannot operate with any assumptions

\(^5\) Side-note: It is instructive to think why is there no natural way of expressing the elimination of existentials other than by analysing the continuations. The connection here is with disjunction. For both of these connectives, introduction takes terms of a subformula/instance of a type—like for products and quantifications; however, the elimination of these takes (modulo the deduction theorem) functions from the subformulas/instances to a type \( T \) to deduce \( T \)—unlike for products where we can deduce (via the projections) their immediate subformulas, and for universal quantifications where we can do direct instantiation.

In the Dual Calculus, where continuations are a first-class concept, these asymmetries disappear.
MENDLER’S INDUCTION OPERATOR

on what \( X \) is. The converse rule, extensionality, is

\[
\text{let } X, x = t \text{ in } h \ (E X. x) =_\eta h t \quad (x, X \text{ not free in } h)
\]

and it essentially says—reading left-to-right\(^6\)—that when applying existential values to functions we always expect them to be generated by the constructor.

§4.6. Existential Restriction Example To see the influence of the type restriction, let \( n : \text{nat} \) and \( \text{suc} : \text{nat} \rightarrow \text{nat} \) be a (Church encoded) natural and the successor function, respectively. The type system correctly—if over-zealously—rejects the following term:

\[
\text{let } X, x = (E \text{nat. } n) \text{ in suc } x.
\]

In typing the rightmost sub-term, we are faced with \( x : X \vdash \text{suc } x : \text{nat} \)—a clear impossibility because type variable \( X \) is not (yet!) instanced to \( \text{nat} \). Were the restriction not there, we would have nonetheless gotten the well-behaved term \( \text{suc } n \).

§4.7. Encapsulation with ADTs Consider the following (suggestive) abstract data type declaration with some (equally suggestive) fields:

\[
\text{data } AMendlerNatStep \text{ where:}
\begin{align*}
\text{aHiddenValue} : & 1 \lor AMendlerNatStep \\
\text{aRecursiveCall} : & AMendlerNatStep \rightarrow \text{Bool}
\end{align*}
\]

(Abstract data types are often composed by a few constructors, some modifiers, and some functions to return meaningful, concrete, and non-abstract, information from the data. The abstract data types we will be interested in shall only have functions that return or use the hidden information.)

Instances (read, elements) of an abstract data type contain the methods present in the declaration—somewhat confusingly, they do not contain any of the underlying data themselves. Instances of the declaration above, then, should be records with two fields: the first will have an optional abstract value (whose type we do not know concretely), and the second is a function whose argument is of that abstract type. Because the type is abstract we can only ever use \( \text{aHiddenValue} \) with two things: polymorphic functions—for they take values of any type, including the hidden one—and the function in the field \( \text{aRecursiveCall} \) which, by construction, has been engineered to accept values of \( AMendlerNatStep \).

§4.8. Typing ADTs With this in mind, the recipe for typing abstract data-types should come as no surprise: each field should correspond to a component of a conjunction of the same type; and, crucially, the name of the abstract data type is a variable which stands for the type of underlying data of the concrete instances—and, thus, it must be existentially quantified. For the example above of \( AMendlerNatStep \) we would have

\[
\text{MendlerNat} \equiv \exists X. (1 \lor X) \land (X \rightarrow \text{Bool}).
\]

\(^6\)As Girard et al. (1989, 10.6.3) remarked when discussing the extensional rule for co-products, “the direction in which the conversion should work is not very clear: the opposite one is in fact much more natural”. 

101
§4.9. Mendler Restriction  The construction above can, of course, be generalised to arbitrary types (read typeschemes). Let $F(X)$ be such a type, and $A$ the intended response type. We define the Mendler restriction of $F(X)$ to $A$ to be:

$$\widehat{F(X)}_A \triangleq \exists X. F(X) \land (X \to A)$$

Despite its name, the restriction is not too restricting when it comes to the dynamics of induction. For, if we have a map function for $F(X)$,

$$\text{map} : \forall X. \forall Y. (X \to Y) \to (F(X) \to F(Y)),$$

and some function $f : F(A) \to A$, we can use them with the restriction by applying

$$\lambda x : \widehat{F(X)}_A. \text{let } X, v = x \text{ in } f \left(\text{map} A (\pi_2 v) (\pi_1 v)\right).$$

Hence, if, by some magic, the hidden function happens to be the inductive invocation of $f$, denoted by $\text{itr}_f : \mu X. F(X) \to A$, and the hidden value is $t : F(\mu X. F(X))$ then applying them to the above would reduce to

$$f \left(\text{map} (\mu X. F(X)) A \ \text{itr}_f t\right),$$

the primitive induction step. The magic we need to make this work is the Mendler induction operator.

§4.10. Mendler Induction Ingredients  Mendler induction is comprised of four things: a (least) fixpoint type constructor, a constructor for instances of the type operator, the associated destructor (the induction operator), and the associated conversion identity. These are all summarised in table 4.3.

§4.11. Inductive Types  An inductive type is (intuitively\footnote{This is the typical case and so we focus on it. However, the reader is advised that an inductive type may refer to itself as the argument of a function, breaking the intuition.}) a type which is formed by iteratively building values from smaller ones. As such, inductive types are formed by appealing to a typescheme $F(X)$, where some distinguished type variable $X$ possibly appears free. Free occurrences of this $X$ represent the points where the type refers to itself—and where those smaller values appear in the construction. The inductive type associated with a typescheme $F(X)$ is represented by $\mu X. F(X)$. Sometimes, when the induction type variable is clear from context, we will abbreviate this to $\mu F$.

§4.12. Constructor  The inductive constructor creates values of inductive type $\mu X. F(X)$ out of terms of type $F(\mu X. F(X))$. Often, such operators are defined as being parametrized directly by terms of the language (e.g. Girard et al., 1989, chap. 7)—indeed, we shall do exactly that later on. However, in keeping with the spirit of the paper by Mendler (1991), we define the constructor as a family of distinguished functions

$$\text{min}^{F(X)} : F(\mu X. F(X)) \to \mu X. F(X),$$

so that for any term $t$ of type $F(\mu X. F(X))$, we have the following derived rule.

$$\Gamma \vdash t : F(\mu X. F(X))$$

$$\Gamma \vdash \text{min}^{F(X)} t : \mu X. F(X)$$
MENDLER’S INDUCTION OPERATOR

Types:

\[ T := \ldots \mid \mu X.F(X) \]

Terms:

\[ t := \ldots \mid \text{min}^{F(X)} \mid \text{mitr}^{F(X)} \]

Typing Rules:

\[ \vdash \text{min}^{F(X)} : F(\mu X.F(X)) \rightarrow \mu X.F(X) \]
\[ \vdash \text{mitr}^{F(X)} : \forall Y. (\hat{F(X)}_Y \rightarrow Y) \rightarrow \mu X.F(X) \rightarrow Y \]

Conversion:

\[ \text{mitr}^{F(X)} A f (\text{min}^{F(X)} t) = f \left( \text{E} (\mu X.F(X)). \langle t, \text{mitr}^{F(X)} A f \rangle \right) \]

Table 4.3: Extension of System F with Mendler induction

§4.13. Eliminator

Conversely, the eliminator (or destructor, or inductor) takes elements from the inductive type to some other type \( A \). To do so, the eliminator must be given some function which constructs an element of type \( A \)—not from an element of \( F(A) \) as in ordinary induction, but—from an element of the Mendler restriction of \( F(X) \):

\[ \text{mitr}^{F(X)} : \forall Y. (\hat{F(X)}_Y \rightarrow Y) \rightarrow \mu X.F(X) \rightarrow Y. \]

We will frequently drop the typescheme annotation when understood from context.


The task of the conversion rule is to construct the relevant values of the type of the Mendler restriction to be passed to the inductive step. Given an inductive call

\[ \text{mitr}^{F(X)} A f (\text{min}^{F(X)} t), \]

we unpack the inductive value and pair it with the inductive call

\[ \langle t, \text{mitr}^{F(X)} A f \rangle; \]

then, we hide the inductive type \[8\], and pass those values to the inductive step,

\[ f \left( \text{E} (\mu X.F(X)). \langle t, \text{mitr}^{F(X)} A f \rangle \right). \]

Here we see how the “magic” step becomes a reality.

\[8\] So that, in particular, we cannot apply the inductive call to the min of the value

\[ (\lambda p : F(\mu X.F(X)) \cdot (\pi_2 p) (\text{min} (\pi_1 p))) \langle t, \text{mitr}^{F(X)} A f \rangle = \text{mitr}^{F(X)} A f (\text{min} t) \]
§4.15. Soundness of Conversion  If this definition is to make any sense we must show, at the very least, that conversion respects the typing structure of the calculus. Modulo the exchange rules, it immediately implies that, if the original calculus is subject preserving, it remains so when augmented with Mendler induction.

Lemma 15. Let \( t \) and \( f \) be two terms such that

\[ \Gamma \vdash \text{mitr}^{F(X)} A f \left( \text{min}^{F(X)} t \right) : A. \]

It therefore follows that

\[ \Gamma \vdash t : F(\mu X. F(X)) \quad \text{and} \quad \Gamma \vdash f : \widehat{F(X)}_A \rightarrow A \]

are derivable; and, thereby,

\[ \Gamma \vdash f(\mathbf{E}(\mu X. F(X)). \langle t, \text{mitr}^{F(X)} A f \rangle) : A \]

is also derivable.

§4.16. Natural Mendler  To cap off this introduction to Mendler’s inductive operators, let us see how to use them to (naïvely) encode the natural numbers. As with the usual form of induction, the base functor will be

\[ \text{NatB}(X) \equiv 1 \lor X. \]

We call it a functor here for we can define a mapping term

\[ \text{NatM} = \Lambda X . \Lambda Y . \lambda f : X \rightarrow Y. [i_1, \lambda x : X. i_2 (f x)] \]

trivially satisfying \( \text{NatM} X Y f (i_1 x) = i_1 x \) and \( \text{NatM} X Y f (i_2 x) = i_2 (f x) \).

We can, thus, define the natural indexed family of terms:

\[ 0 \equiv \text{min} (i_1 \ast) \quad n + 1 \equiv \text{min} (i_2 \bar{n}) \]

If we take some term \( f : F(A) \rightarrow A \), we can define

\[ f^* \equiv \left( \lambda x : \widehat{F(X)}_A. \text{let} X, p = x \text{ in } f \left( \text{NatM} X A (\pi_2 p) (\pi_1 p) \right) \right) \]

and then use an inductive argument to see that, first:

\[
\begin{align*}
\text{mitr} A f^* \tilde{0} &= \text{mitr} A f^* \left( \text{min} \ (i_1 \ast) \right) \\
&= f^* \left( \mathbf{E} \mu X. \text{NatB}(X). \langle i_1 \ast, \text{mitr} A f^* \rangle \right) \\
&= \text{let} X, p = \mathbf{E} \mu X. \text{NatB}(X). \langle i_1 \ast, \text{mitr} A f^* \rangle \text{ in } f \left( \text{NatM} X A (\pi_2 p) (\pi_1 p) \right) \\
&= f \left( \text{NatM} X A (\pi_2 \langle i_1 \ast, \text{mitr} A f^* \rangle) (\pi_1 \langle i_1 \ast, \text{mitr} A f^* \rangle) \right) \\
&= f \left( \text{NatM} X A (\text{mitr} A f^*)(i_1 \ast) \right) \\
&= f \ (i_1 \ast).
\end{align*}
\]
\[ F(X) = \forall Y. Y \to Y \quad G(X) = \forall Y. Y \to X \]
\[ H(X) = \forall Y. (X \land Y) \to Z \quad J(X) = \forall Y. (X \land Y) \to X \]

Table 4.4: Two positive, one negative, and one mixed typeschemes (top to bottom).

Now, for a successor value

\[
mitr \ A f^* \ n + 1 \\
= mitr \ A f^* \ (min \ (i_2 \overline{n})) \\
= f^* \ (E \mu X. \text{NatB}(X). \langle i_2 \overline{n}, mitr \ A f^* \rangle) \\
= \text{let } X, p = E \mu X. \text{NatB}(X). \langle i_2 \overline{n}, mitr \ A f^* \rangle \text{ in } f \ (\text{NatM} \ X \ A \ (\pi_2 \ p) \ (\pi_1 \ p)) \\
= f \ (\text{NatM} \ X \ A \ (\text{mitr} \ A f^*) \ (i_2 \overline{n})) \\
= f \ (i_2 \ (\text{mitr} \ A f^* \overline{n})) \\
= f \ (i_2 \ (f \ (i_2 \ ((f \ (i_1 \ *)) \ldots)))))^{n \ \text{times}} \\
= f \ (i_2 \ (f \ (i_2 \ ((f \ (i_1 \ *)) \ldots)))))^{n+1 \ \text{times}} \\
\]

This is exactly the behaviour one would get by applying \( f \) to the Church encoding of a natural number.

4.2 Negative Typeschemes and Mendler Induction

\[\text{§4.17. Variance of Typeschemes} \quad \text{One of the most intriguing aspects of Mendler induction is its ability to be parametrised by typeschemes which are mixed or even purely negative—note how we made no assumptions about the typeschemes that parametrise the inductive operators. Recall that a positive occurrence of a type variable \( X \) in a type \( T \) is one which occurs to the left of an even number of implications; conversely, a negative occurrence is one which happens to the left of an odd number of implications. Accordingly, a type scheme \( F(X) \) is negative in \( X \) if \( X \) appears only negatively in \( F(X) \); if \( X \) appears both negatively and positively then we say the typescheme is mixed; otherwise, we say it is positive.}\]

\[\text{§4.18. Variance and Functoriality} \quad \text{For ordinary induction, the limiting factor is the definition of a functorial action: There is simply no natural way of defining a co-variant (read, positive) action on morphisms whenever there are negative occurrences of \( X \). The definition of Mendler induction, however, makes no mention of a functorial action. If} \]
such an action exists then it can be used to write an inductive definition; if it doesn’t, then no harm arises.

§4.19. Example: a \( \lambda \)-calculus  One might be led to think that this ability to handle non-positive typeschemes is vacuous in that it is impossible to come up with functions that can consume values in meaningful ways—and for strictly negative typeschemes that would be correct. For typeschemes of mixed variance, however, this extra facility can be put to good use. Consider the implementation of a simple \( \lambda \)-calculus with a distinguished term in a higher-order abstract syntax\(^9\) fashion (Pfenning and Elliott, 1988) as the least fix-point type \( \mu_A \) of the typescheme

\[
\Lambda(X) = 1 \lor (X \to X) \lor (X \land X).
\]

The disjuncts represent, respectively, the constant, abstractions and applications.

A couple of examples: the identity function can written as

\[
\min^\Lambda (t_2 (\lambda x : \mu_A . x));
\]

the self-application function, \( \lambda x . x \ x \), can be written as:

\[
\min^\Lambda (t_2 (\lambda x : \mu_A . \min^\Lambda (t_3 (x, x)))).
\]

§4.20. Example: Testing for Normal Forms  Whilst we cannot write an evaluation function for this calculus—which would break strong-normalization—we can ask of a “closed term”\(^10\) whether it is a normal form or not. By this we understand a term which is either an abstraction or an application where the term on the left is not an abstraction and the one on the right is a normal form. We shall represent these three cases by the type \( T = 1 \lor 1 \lor 1 \); the three injections will represent, respectively, the case where we do not have normal form \( (t_1) \), the case where we have a (necessarily normal) lambda abstraction \( (t_2) \), and the case where we have an applicative normal form \( (t_3) \). The overall shape of the answer we are looking for must be something like

\[
f = \lambda ex : \Lambda(X)_T . \text{let } X, p = ex \text{ in } [\text{caseConst, caseLambda, caseApp}] (\pi_1 p)
\]

The constant (a normal form) and abstraction cases boil down to applying the respective injections as needed:

- \( \text{caseConst} \equiv t_3 \) : \( 1 \to T \)
- \( \text{caseLambda} \equiv \lambda t : X \to X . t_2 \ast \) : \( (X \to X) \to T \).

The application case begins by applying the function on the hidden type (read, the induction) to the first value (read, the left sub-term). If this returns a non-normal form or an abstraction, we’re done—not a normal form; otherwise, it all depends on whether the value on the right is a normal form or not. Abbreviating the function \( \pi_2 p \) on the hidden type by \( \rho \):

\[
\text{caseApp} \equiv \lambda v : X \times X . [t_1, t_1, \lambda t : T . [t_1, t_3, t_3] (\rho (\pi_2 v)) (\rho (\pi_1 v))) : X \times X \to T.
\]

\(^9\)Ahn and Sheard (2011) offer a much more elaborate implementation using higher-order types.

\(^10\)In higher-order abstract syntax we don’t really have direct access to variables.
MENDLER AND ORDINARY INDUCTION

Types:
\[ T = \ldots \mid \mu X.F(X) \quad \text{(for positive } F(X)) \]

Terms:
\[ t = \ldots \mid in^{F(X)} \mid itr^{F(X)} \]

Typing Rules:
\[ \vdash in^{F(X)} : F(\mu X.F(X)) \rightarrow \mu X.F(X) \]
\[ \vdash itr^{F(X)} : \forall X.(F(Y) \rightarrow Y) \rightarrow \mu X.F(X) \rightarrow Y \]

Conversion:
\[ itr^{F(X)} A f (in^{F(X)} t) = f \left( F \mu F A (itr^{F(X)} A f) t \right) \]
\[ \text{(where } F = \Lambda X . \Lambda Y . \lambda f : X \rightarrow Y . \lambda x : F(X). \text{ map}^{F(X)}[f,x]) \]

Table 4.5: Extension of System F with ordinary induction

4.3 Mendler and Ordinary Induction

§4.21. Relating the Two Inductions  After introducing Mendler induction, we are now in condition to relate it precisely to ordinary induction. Thus far we have treated ordinary induction rather intuitively; to achieve formal results, we must right that wrong. The extension of System F with induction operators is given in table 4.5.

§4.22. Functors & Maps  Given a positive typescheme \( F(X) \), we need a way to transform some instance \( F(T) \) into another \( F(U) \) provided that we know how to transform elements of type \( T \) into elements of \( U \). This is used in the conversion of induction to transform a term of type \( F(\mu X.F(X)) \) into one of type \( F(A) \) by using an inductive function from \( \mu X.F(X) \) to \( A \). The definition of such a term is defined by induction on the definition of the typescheme \( F(X) \) in table 4.6. Note that in a positive typescheme \( F(X) = G(X) \rightarrow H(X) \), \( X \) may (only) appear negatively in \( G(X) \); in this case, the action reverts the order, taking elements of \( F(U) \) into elements of \( F(T) \).

Given one such term we can define a closed term which encodes the functorial action by simply
\[ F = \Lambda X . \Lambda Y . \lambda f : X \rightarrow Y . \lambda x : F(X). \text{ map}^{F(X)}[f,x] \]

Abusing notation, we will assume the variable \( X \) is understood from context and drop it altogether. The following proposition asserts that our claim pertaining the type of the functorial action—in particular what happens when we go to the left of an implication sign—is correct.

\[ ^{11} \text{We could have just as well defined the map for a typescheme as a closed term, although this complicates the term a little because the instantiations need to be passed down to the recursive invocations.} \]
MENDLER INDUCTION

\[
\begin{align*}
\text{map}^{Z(X)}[f, t] &= t \\
\text{map}^{X(X)}[f, t] &= f t \\
\text{map}^{G(X)\rightarrow H(X)}[f, t] &= \lambda x : G(X). \text{map}^{H(X)}[f, t \ \text{map}^{G(X)}[f, x]] \\
\text{map}^{G(X)\wedge H(X)}[f, t] &= \langle \text{map}^{G(X)}[f, \pi_1 t], \text{map}^{H(X)}[f, \pi_2 t] \rangle \\
\text{map}^{\exists Z.F(X)}[f, t] &= \Lambda Z . \ \text{map}^{F(X)}[f, t \ Z] \\
\text{map}^{\mu Z.F(X)}[f, t] &= \text{let } Z, \ z = t \ \text{in} Z. \ \text{map}^{F(X)}[f, z] \\
\text{map}^{\mu Z.F(X) Z}[f, t] &= \text{itr}^{F(X)(Z)}(\mu Z.F(Y)(Z)) \\
&\quad (\lambda z : F(Y)(\mu Z . F(X)(Z)). \text{in}^{F(Y)(Z)} \ \text{map}^{F(X)}[f, z]) t
\end{align*}
\]

Table 4.6: The functorial actions for \( F(X) \) for terms \( f \) and \( t \).

**Proposition 12** (Functorial Action). Let \( \Gamma \vdash f : T \rightarrow S \); we have that

- **F positive and** \( \Gamma \vdash t : F(T) \implies \Gamma \vdash \text{map}^{F(X)}[f, t] : F(S) \);
- **F negative and** \( \Gamma \vdash t : F(S) \implies \Gamma \vdash \text{map}^{F(X)}[f, t] : F(T) \).

**Proof.** We proceed by mutual induction on the types parametrizing the actions, generalizing on both the context and the terms. We focus only on the inductive case.

\[
\begin{align*}
\Gamma, z : F(T)(\mu Z.F(S)(Z)) \vdash z : F(T)(\mu Z.F(S)(Z)) & \quad \text{([IH])} \\
\Gamma, z : F(T)(\mu Z.F(S)(Z)) \vdash \text{map}^{F(X)}[f, z] : F(S)(\mu Z.F(S)(Z)) & \\
\Gamma, z : F(T)(\mu Z.F(S)(Z)) \vdash \text{in}^{F(S)(Z)} \ \text{map}^{F(X)}[f, z] : \mu Z.F(S)(Z) \\
\Gamma \vdash \lambda z : F(T)(\mu Z . F(S)(Z)). \text{in}^{F(S)(Z)} \ \text{map}^{F(X)}[f, z] : F(T)(\mu Z.F(S)(Z)) \rightarrow \mu Z.F(S)(Z) \\
\Gamma \vdash \text{itr}^{F(T)(Z)}(\mu Z.F(S)(Z)) (\lambda z : \ldots) : \mu Z.F(S)(Z) \\
\Gamma \vdash \text{map}^{\mu Z.F(X)(Z)}[f, t] : \mu Z.F(S)(Z) \\
\Gamma, z : F(S)(\mu Z.F(T)(Z)) \vdash z : F(S)(\mu Z.F(T)(Z)) & \quad \text{([IH])} \\
\Gamma, z : F(S)(\mu Z.F(T)(Z)) \vdash \text{map}^{F(X)}[f, z] : F(T)(\mu Z.F(T)(Z)) \\
\Gamma, z : F(S)(\mu Z.F(T)(Z)) \vdash \text{itr}^{F(T)(Z)} \ \text{map}^{F(X)}[f, z] : \mu Z.F(T)(Z) \\
\Gamma \vdash \lambda z : F(S)(\mu Z . F(T)(Z)). \text{in}^{F(T)(Z)} \ \text{map}^{F(X)}[f, z] : F(S)(\mu Z.F(T)(Z)) \rightarrow \mu Z.F(T)(Z) \\
\Gamma \vdash \text{itr}^{F(S)(Z)}(\mu Z.F(T)(Z)) (\lambda z : \ldots) : \mu Z.F(T)(Z) \\
\Gamma \vdash \text{map}^{\mu Z.F(X)(Z)}[f, t] : \mu Z.F(T)(Z)
\end{align*}
\]

\[\square\]

**Corollary 4.** For any positive typescheme \( F \), the associated closed functorial action satisfies

\[\vdash F : \forall X . \forall Y . (X \rightarrow Y) \rightarrow F(X) \rightarrow F(Y) .\]
MENDLER AND ORDINARY INDUCTION

Proof. Follows easily from proposition \[12\]. The abstraction side-condition is trivially satisfied as it is only ever applied to closed terms.

\[ f : X \rightarrow Y, x : F(X) \vdash map^{F(X)}[f, x] : F(Y) \]

\[ f : X \rightarrow Y \vdash \lambda x : F(X). map^{F(X)}[f, x] : F(X) \rightarrow F(Y) \]

\[ \vdash \lambda f : X \rightarrow Y. \lambda x : F(X). map^{F(X)}[f, x] : (X \rightarrow Y) \rightarrow F(X) \rightarrow F(Y) \]

\[ \vdash \Lambda X. \Lambda Y. \lambda f : X \rightarrow Y. \lambda x : F(X). map^{F(X)}[f, x] : \forall X. \forall Y. (X \rightarrow Y) \rightarrow F(X) \rightarrow F(Y) \]

\[ \Box \]

§4.23. Correctness for Inductive Types We can now show that the definition is not senseless—as usual, by stating that the resulting system preserves typings.

Theorem 15. Assuming the base calculus is subject preserving, so is its extension with ordinary induction.

Proof. In any context \( \Gamma \), for well-typed \( \text{itr}^{F(X)} A f \ (\text{in}^{F(X)}) t \), we must have

\[ \Gamma \vdash f : F(A) \rightarrow A \quad \text{and} \quad \Gamma \vdash t : F(\mu X.F(X)). \]

Therefore, it is easy to see,

\[ \Gamma \vdash \text{itr}^{F(X)} A f : \mu F \rightarrow A; \]

\[ \Gamma \vdash F \mu F A (\text{itr}^{F(X)} A f) : F(\mu F) \rightarrow F(A) \]

\[ \Gamma \vdash F \mu F A (\text{itr}^{F(X)} A f) t : F(A) \]

\[ \Gamma \vdash f (F \mu F A (\text{itr}^{F(X)} A f) t) : A \]

\[ \Box \]

§4.24. Induction via Mendler We have already hinted at how ordinary induction can be performed using Mendler induction (§4.8). The crux of the idea is that ordinary induction passes the result of the inductive invocations to the induction step whereas Mendler induction passes the inductive invocation itself; it stands to reason, then, that, given an (ordinary) inductive step, all we have to do to simulate ordinary induction using Mendler induction is to simply compose \( f \) with the functorial action applied to the (hidden) inductive invocation. The translation based on this idea for the implicatinal fragment of System F with induction is defined in table \[4.7\]. Apart from the aforementioned case, the definition is straightforward: we just confuse the types in both systems and most terms are handled by homomorphism. Note that, because translation for an iteration \( \text{itr}^{F(X)} \) requires translating the associated functorial action, the definition is not by induction on terms but by induction on terms and on the typescheme annotations in \( \text{itr} \).

Proposition 13. For any well-typed term of the implicatinal fragment of System F with ordinary induction \( \Gamma \vdash t : T \), its translation with Mendler induction is well-typed, \( \Gamma \vdash T : T \). This because we also have

\[ \vdash F : \forall X. \forall Y. (X \rightarrow Y) \rightarrow F(X) \rightarrow F(Y) \]

for any positive typescheme \( F(X) \).

Proof. By mutual course-of-values induction on terms and annotation of iterations. \[ \Box \]
MENDEL INDUCTION

Types:
\[ \mathcal{T} = T \]

Terms:
\[ \bar{x} = x \]
\[ \lambda x : \mathcal{T}, t = \lambda x : \mathcal{T}, \bar{t} \]
\[ \Lambda X : t = \Lambda X : \bar{t} \]
\[ \mathit{in}^{F(X)} = \mathit{min}^{F(X)} \]
\[ \mathit{itr}^{F(X)} = \Lambda Y \cdot \lambda f : F(Y) \to Y, \mathit{mitr}^{F(X)} Y \left( \lambda e : \widehat{F(Y)}_Y \cdot \text{let } Z, z = e \\text{ in } f (\overline{P} Z Y (\pi_2 z) (\pi_1 z)) \right) \]

Table 4.7: The translation of the implicational fragment with ordinary induction into System F with Mendler induction

§4.25. Modelling Induction  Adapting the terminology of the study of programming languages, the previous proposition takes care of the adequacy of the translation when it comes to the statics of the induction, proving that types—a static property of the calculus—are preserved. To prove that the dynamics of the calculus are adequately modelled by the translation we require a few more results about the main vehicle for conversion: substitution. We must show that substitution preserves the translation. One wrinkle here is that when we substitute by a type in a typescheme, the associated map function changes.

Lemma 16. Let \( T(X) \) be type \( T \) viewed as a (constant) typescheme—i.e. in which \( X \) is not present; the mapping function is the identity: for any \( A, B, \) and \( f : A \to B, \)

\[ \mathit{map}^{T(X)}[f, t] = t \]

Proof. By induction on the definition of the mapping function and through extensive use of extensional equality for the different types. \( \square \)

Theorem 16. Given \( F(X) \) a typescheme, \( Y \neq X, T \) a type, and a term \( t = t[T/X] \) we have that

\[ \mathit{map}^{F(X)}[f, t][T/X] = \mathit{map}^{F[T/Y]}[(X)[f, t[T/X]]}. \]

Consequently,

\[ F_X[T/Y] = (F[T/Y])_X \]

Lemma 17 (Substitution). The different notions of substitution are preserved by the translation; for \( T, U \) types, \( X \) and \( Y \) types variables (as usual, assumed not to be bound in the relevant types), and \( t \) and \( u \) terms:

1. \( U[T/X] = U[T/X] \)
2. \( u[t/x] = u[t/x] \)
MENDLER AND ORDINARY INDUCTION

3. $u[T/X] = \overline{\pi[T/X]}$

Proof. Each point is proven by induction: point 1 on the structure of types; point 2 induction on terms (note that for the inductor case, the translation is a closed term and the substitution has no effect); and point 3 is proven by induction on the term structure and on the type annotations for iterator. For this last point, crucially, we have for the iterator (and modulo the type annotations) that

$$\overline{itr[F,T/X]} = \left( \Lambda W . \lambda f : \ldots . \text{mitr} W \left( \lambda e : \ldots . \text{let } Z, z = e \text{ in } f \left( F Z W (\pi_2 z) (\pi_1 z) \right) \right) \right) [T/X]$$

$$= \left( \Lambda W . \lambda f : \ldots . \text{mitr} W \left( \lambda e : \ldots . \text{let } Z, z = e \text{ in } f \left( F[T/X] Z W (\pi_2 z) (\pi_1 z) \right) \right) \right)$$

hence, by the induction hypothesis,

$$= \left( \Lambda W . \lambda f : \ldots . \text{mitr} W \left( \lambda e : \ldots . \text{let } Z, z = e \text{ in } f \left( F[T/X] \right) Z W (\pi_2 z) (\pi_1 z) \right) \right)$$

$$= \text{itr}^{F[T/X]}[Y]$$

$$= \text{itr}^{F[T/X]}[T/X]$$

$$\Box$$

Proposition 14 (Homomorphism). The translation of any two convertible terms $t$ and $u$ of System F with induction into the System F are also convertible:

$$t = u \implies \overline{t} = \overline{u}$$

Proof. A simple case analysis on the definition of $\beta$-conversion suffices. Focusing on the inductive case:

$$\overline{itr[F,X] A f \left( \text{in}^{F[X]} t \right)}$$

$$= \overline{itr[F,X] A f \left( \text{in}^{F[X]} t \right)}$$

$$= \text{mitr}^{F[X] A} \left( \lambda e : \overline{F[X] A} . \text{let } z, Z = e \text{ in } \overline{f} (\overline{T Z A} (\pi_2 z) (\pi_1 z)) \right) \left( \text{min}^{F[X]} \overline{t} \right)$$

$$= \left( \lambda e : \overline{F[X] A} . \text{let } z, Z = e \text{ in } \overline{f} (\overline{T Z A} (\pi_2 z) (\pi_1 z)) \right)$$

$$= \left( E \mu X. F(X). \left( \overline{t} . \text{mitr}^{F[X]} Y \left( \lambda e : \overline{F[X] A} . \text{let } z, Z = e \text{ in } \overline{f} (\overline{T Z A} (\pi_2 z) (\pi_1 z)) \right) \right) \right)$$

111
§4.26. Mendler Induction Ordinarily  Surprisingly, there is a construction that yields Mendler induction from ordinary induction—provided that the type system is expressive enough \cite{Uustalu:1999}. Recall the definition of the Mendler iterator:
\[
mitr^{F(X)} : \forall Y . \left( \overline{F(X)}_Y \rightarrow Y \right) \rightarrow \mu X. F(X) \rightarrow Y.
\]
If we fix the return type \( A \) the iterating function \( f \) must have type
\[
f : \overline{F(X)}_A \rightarrow A.
\]
This suggests that \( f \) is an algebra—to borrow the notion from Category Theory—not for \( F(X) \) but for \( \overline{F(X)}_A \), seen as a functor on \( A \). It will turn out, that the initial algebra for this functor—i.e. its primitive inductor—behaves similarly to the Mendler inductor.

§4.27. Functoriality of \( \overline{F(X)}_A \)  Of course, we need to show that \( \overline{F(X)}_A \) behaves in some reasonable sense like a functor. The action on types is, we recall, given by
\[
\overline{F(X)}_Y \equiv \exists X. F(X) \land (X \rightarrow Y).
\]
We assume that \( Y \) is not free in \( F(X) \); as such, \( Y \) appears only positively in \( \overline{F(X)}_Y \). The action on morphisms follows naturally by composition: for \( \Gamma \vdash f : A \rightarrow B \),
\[
\overline{F(X)}_f \equiv \lambda t : \overline{F(X)}_A . \text{ let } X, x = t \text{ in } E X . \langle \pi_1 x, \lambda y : X . f ((\pi_2 x) y) \rangle.
\]
Its type under \( \Gamma \) is \( \overline{F(X)}_A \rightarrow \overline{F(X)}_B \). It is easy to see that identities are preserved
\[
\overline{F(X)}_f = \lambda t : \overline{F(X)}_A . \text{ let } X, x = t \text{ in } E X . \langle \pi_1 x, \lambda y : X . (\pi_2 x) y \rangle
\]
\[
\overline{F(X)}_f = \lambda t : \overline{F(X)}_A . \text{ let } X, x = t \text{ in } E X . \langle \pi_1 x, \lambda y : X . (\pi_2 x) y \rangle
\]
\[
\overline{F(X)}_f = \lambda t : \overline{F(X)}_A . \text{ let } X, x = t \text{ in } E X . \langle \pi_1 x, \lambda y : X . (\pi_2 x) y \rangle
\]
\[
\overline{F(X)}_f = \lambda t : \overline{F(X)}_A . \text{ let } X, x = t \text{ in } E X . \langle \pi_1 x, \lambda y : X . (\pi_2 x) y \rangle
\]
\[
\overline{F(X)}_f = \lambda t : \overline{F(X)}_A . \text{ let } X, x = t \text{ in } E X \times X
\]
MENDLER AND ORDINARY INDUCTION

For composition we need a (fusion) lemma for existentials; in the equality

$$
\lambda t : \widehat{F(X)}_A \cdot \widehat{F(X)}_f \left( \widehat{F(X)}_{\delta} t \right) = \widehat{F(X)}_{fog}
$$

the left-hand side is comprised of two unpacking terms, the right-hand side of only one. Note then, that for arbitrary \( h, t, \) and \( t' \),

$$
h \left( \text{let } X, x = t \text{ in } t' \right)
= (\lambda z : \exists X . T . h \left( \text{let } X, x = z \text{ in } t' \right)) t
= \text{let } Y, y = t \text{ in } (\lambda z : \exists X . T . h \left( \text{let } X, x = z \text{ in } t' \right)) E Y. y
= \text{let } Y, y = t \text{ in } h \left( \text{let } X, x = E Y. y \text{ in } t' \right)
= \text{let } Y, y = t \text{ in } h \left( t'[Y/X] \right[ y/x \left) \right]
= \text{let } X, x = t \text{ in } h t'
$$

where the auxiliary variables are fresh and the last step is justified by \( \alpha \)-equivalence. The result we actually need is an instance of this: take \( t, t' \), and \( t'' \) to be three terms such that \( x \) and \( X \) are not free in \( t \) and for which \( z \) is fresh; then

$$
\text{let } X, x = (\text{let } Y, y = t \text{ in } E X. t') \text{ in } E X. t''
= (\lambda z : \exists X . T . \text{let } X, x = z \text{ in } E X. t'') \left( \text{let } Y, y = t \text{ in } E Y. t' \right)
= \text{let } Y, y = t \text{ in } (\text{let } X, x = E Y'. t' \text{ in } E X. t'')
$$

And now we can prove that composition is preserved:

$$
\lambda t : \widehat{F(X)}_A \cdot \widehat{F(X)}_f \left( \widehat{F(X)}_{\delta} t \right)
= \lambda t : \widehat{F(X)}_A \cdot (\text{let } X, x = t \text{ in } E X. \langle \pi_1 x, \lambda y : X . f ((\pi_2 x) y) \rangle) \left( \widehat{F(X)}_{\delta} t \right)
= \lambda t : \widehat{F(X)}_A \cdot \text{let } X, x = \left( \widehat{F(X)}_{\delta} t \right) \text{ in } E X. \langle \pi_1 x, \lambda y : X . f ((\pi_2 x) y) \rangle
= \lambda t : \widehat{F(X)}_A \cdot \left( \text{let } X, x = (\text{let } Y, y = t \text{ in } E Y. \langle \pi_1 y, g (\pi_2 y) \rangle) \right) \left( \widehat{F(X)}_{\delta} t \right)
= \lambda t : \widehat{F(X)}_A \cdot \text{let } Y, y = t \text{ in } (\text{let } X, x = E Y. \langle \pi_1 y, g (\pi_2 x) \rangle) \text{ in } E X. \langle \pi_1 x, f (\pi_2 x) \rangle)
= \lambda t : \widehat{F(X)}_A \cdot \text{let } Y, y = t \text{ in } E X. \langle \pi_1 y, f (g (\pi_2 x)) \rangle
= \lambda t : \widehat{F(X)}_A \cdot \text{let } Y, y = t \text{ in } E X. \langle \pi_1 y, (\lambda z : A . f (g z)) (\pi_2 x) \rangle)
= \widehat{F(X)}_{fog}
$$

§4.28. The Translation of Mendler Induction  The embedding of Mendler induction into the system with ordinary induction is completely determined by our choice of interpreting the Mendler inductive closure of \( F(X) \) by the inductive closure of \( \widehat{F(Y)}_X \): the translation for types (table 38) corresponds to the homomorphic closure under this one rule. This choice requires some housekeeping as a consequence of doing iteration over the more complex type. First, given a typescheme \( F(X) \), we have posited that the translation
MENDLER INDUCTION

\[ |X|^T = X \]
\[ |A \land B|^T = |A|^T \land |B|^T \]
\[ |A \rightarrow B|^T = |A|^T \rightarrow |B|^T \]
\[ |\forall X \cdot A|^T = \forall X \cdot |A|^T \]
\[ |\exists X \cdot A|^T = \exists X \cdot |A|^T \]
\[ |\mu X . F(X)|^T = \mu X . |F|^T(\overline{Y}_X) \quad (Y \text{ fresh}) \]

Table 4.8: Translation of types in System F with Mendler induction to System F with ordinary induction

of its Mendler inductive least fix-point is to be given by \( \mu X . |F|^T(\overline{Y}_X) \) (cf. table 4.8). Algebras for this functor have the same type as the translated Mendler induction step. The translation, then, simply maps the Mendler iterator to the usual iterator (cf. table 4.9). On the other hand, the inductor, \( \min : F(\mu X . F(X)) \rightarrow \mu X . F(X) \), must be now of type

\[ \overline{F}^T(X)_{\mu X . |F|^T(\overline{Y}_X)} \rightarrow \mu X . |F|^T(\overline{Y}_X), \]

i.e.,

\[ \exists X . |F|^T(X) \times (X \rightarrow \mu X . |F|^T(\overline{Y}_X)) \rightarrow \mu X . |F|^T(X)_X. \]

However, the translation of \( F(\mu X . F(X)) \) is

\[ |F|^T\left(\mu X . |F|^T(\overline{Y}_X)\right) \]

and so we need to adapt the term received by \( \text{in} \) to the former type. The latter type conforms with the type of the abstracted value in the definition of the Mendler type restriction. Using that as our value of type \( |F|^T(X) \) entails that we also have to give a function of type

\[ \mu X . |F|^T(\overline{Y}_X) \rightarrow \mu X . |F|^T(\overline{Y}_X). \]

The identity suggests itself.

§4.29. Simulating Mendler Induction To see why the identity is not only suggestive but necessary, let us analyse abstractly the mechanics of how the encoding simulates Mendler induction. An iteration \( \text{itr}^{F(X)} f \) converts into

\[ f \left( F \left( \text{itr}^{F(X)} f \right) \right). \]

For the functor associated with Mendler induction, \( \overline{F(X)}_Y \), the action of the functor simply pre-composes the argument function with the hidden function. But pre-composing the iteration with the identity in the hidden function results in iteration being captured and
MENDLER AND ORDINARY INDUCTION

\[ |x|' = x \]
\[ |\langle t, t' \rangle|' = \langle |t|', |t'|' \rangle \]
\[ |\lambda x : T . t|' = \lambda x : |T|^T . |t|' \]
\[ |\Lambda X . t|' = \Lambda X . |t|' \]
\[ |E T . t|' = E |T|^T . |t|' \]
\[ |\text{min}^{F(X)}|' = \lambda x : |F|^T(X)_{\mu X.F(X)^T} \cdot \text{in}^{F(T)(X)}_{\mu X.F(X)^T} (E |\mu F|^T . \langle x, y : |\mu X.F(X)|^T . y \rangle) \]
\[ |\text{mitr}^{F(X)}|' = \text{itr}^{F(T)(X)}_{\mu X.F(X)^T} \]

Table 4.9: Translation of terms in System F with Mendler induction to System F with ordinary induction

passed to \( f \). At first blush surprising, the proof that Mendler induction can be simulated by ordinary induction is actually simpler than the converse; in hindsight, though, Mendler induction relies on less structure than ordinary induction—and hence proofs follow more naturally.

**Proposition 15** (Soundness of the Translation). *The translation preserves typings:*

\[ \Gamma \vdash t : T \implies |\Gamma|^T \vdash |t|' : |T|^T \]

**Proof.** We reason by induction on the terms. The only interesting case is that of the injection, which is responsible for the heavy-lifting of the induction.

\[ x : |F|^T(\langle x, \mu X.F(X) \rangle^T), |\Gamma|^T \vdash x : |F|^T(\langle \mu X.F(X) \rangle^T) \quad \text{(trivial for id)} \]
\[ x : |F|^T(\langle \mu X.F(X) \rangle^T), |\Gamma|^T \vdash x, |\text{id}|^T : |F|^T(\langle \mu X.F(X) \rangle^T) \land (\langle \mu X.F(X) \rangle^T \rightarrow |\mu X.F(X)|^T) \]
\[ x : |F|^T(\langle \mu X.F(X) \rangle^T), |\Gamma|^T \vdash x, |\text{id}|^T : |F|^T(\langle x, \mu X.F(X) \rangle^T) \land (\langle \mu X.F(X) \rangle^T \rightarrow |\mu X.F(X)|^T) \]
\[ x : |F|^T(\langle \mu X.F(X) \rangle^T), |\Gamma|^T \vdash x, |\text{id}|^T : |F|^T(\langle x, \mu X.F(X) \rangle^T) \land (\langle \mu X.F(X) \rangle^T \rightarrow |\mu X.F(X)|^T) \]
\[ |\Gamma|^T \vdash \lambda x : |F|^T(\langle x, \mu X.F(X) \rangle^T). |\mu X.F(X)|^T . \langle x, |\text{id}|^T \rangle : |F|^T(\langle \mu X.F(X) \rangle^T) \]

**Lemma 18** (Substitution). *Substitution preserves the interpretation of Mendler induction into System F with induction. For \( T, U \) types, \( X \) and \( Y \) types variables, and \( t \) and \( u \) terms:*

1. \[ |U[T/X]|^T = |U|^T[|T|^T/X] \]
2. \[ |t[T/X]|' = |t'|'[|T|^T/X] \]
3. \(|t[t/x]|^t = |t'|[|t'|/x]|

Proof. The proof for the first property is by induction on types; for the succeeding ones, it is by induction on the structure of terms—details are omitted. □

Theorem 17. The translation is a homomorphism of β-conversions:

\(t = t' \implies |t'| = |t'|^t\)

Proof. Standard case analysis of β-conversion. Focusing on the most elaborate of all cases, induction, we get

\[
|mtr^{F(X)} A f (\text{min}^{F(X)} t)|^t
= \text{itr}^{F(X)} |A|^T |f|^t \left( \text{in}^{F(X)} |E| \left( |\mu X.F(X)|^T . \left( |t'|, |id|^t \right) \right) \right)
= |f|^t \left( |F|^T (Y) \text{itr}^{F(X)} |A|^T |f|^t \left( |E| \left( |\mu X.F(X)|^T . \left( |t'|, |id|^t \right) \right) \right) \right)
= |f|^t \left( \left( |E| \left( |\mu X.F(X)|^T . \left( |t'|, \lambda y : |\mu X.F(X)|^T . \text{itr}^{F(X)} |A|^T |f|^t (|id|^t y) \right) \right) \right)
= |f|^t \left( \left( |E| \left( |\mu X.F(X)|^T . \left( |t'|, \lambda y : |\mu X.F(X)|^T . \text{itr}^{F(X)} |A|^T |f|^t y \right) \right) \right)
= |f|^t \left( \left( |E| \left( |\mu X.F(X)|^T . \left( |t'|, \lambda y : |\mu X.F(X)|^T . \text{itr}^{F(X)} |A|^T |f|^t y \right) \right) \right)
= |f|^t \left( \left( |E| \left( |\mu X.F(X)|^T . \left( |t'|, \lambda y : |\mu X.F(X)|^T . \text{itr}^{F(X)} |A|^T |f|^t y \right) \right) \right)
= |f|^t \left( \left( |E| \left( |\mu X.F(X)|^T . \left( |t'|, mtr^{F(X)} A f \right) \right) \right) \right)
= |f|^t \left( \left( |E| \left( |\mu X.F(X)|^T . \left( |t|, mtr^{F(X)} A f \right) \right) \right) \right)
= |f|^t \left( |E| \left( |\mu X.F(X)|^T . \left( |t, mtr^{F(X)} A f \right) \right) \right)
= |f| \left( |E| \left( |\mu X.F(X)| . \left( |t, mtr^{F(X)} A f \right) \right) \right)
= |f| \left( \left( |E| \left( |\mu X.F(X)| . \left( |t, mtr^{F(X)} A f \right) \right) \right) \right)

□
Chapter 5

Mendler Induction in the Dual Calculus

§5.1. “And Two Become One” So far we have surveyed the Dual Calculus and Mendler induction individually; now is time to put the two together. Our base calculus will be the full second-order Dual Calculus with implication and subtractive types studied in chapter 2 (tables 2.20 and 2.21). To this we will add the appropriate inductive types, terms and co-term constructors, and reduction rule. Concomitantly, we will also add co-induction—this is done by straight dualization to the point where it comes essentially “for free”.

In the following chapter we show that this system is well-behaved—it is subject preserving, self-dualizing, and strongly-normalizing. Yet, here we show that it is also very expressive: as a warm-up exercise we will see how to encode natural numbers and express their sum inductively; more generally, up to some mild assumptions, we can encode the Intuitionistic version of Mendler induction within this Classical interpretation; even further, we will see that Classical Mendler induction subsumes Classical ordinary induction as introduced by Kimura and Tatsuta (2009).

5.1 Mendler Induction and the Dual Calculus

§5.2. The Parametrized Mendler Operators It is well-known that abstractions, both on the term and type level, which are harmless for Intuitionistic languages such as the lambda calculus, interfere awkwardly with the control features found in classical languages such as DC (Parigot, 2000, p. 446); further, if not properly considered, they may even create very naughty systems, indeed (Harper and Lillibridge, 1991). To lessen our dependency on both kinds of functional types (on the type and term level), we shall model the constructor and the destructor as operators that are explicitly parametrized by the (co-) variables, and phrases they operate on. We say ‘lessen’, not eliminate, for reasons that we will become apparent when we deal with the eliminator for inductive types/introduction for co-inductive types. With all this in mind, the extension of second-order DC with Mendler recursion in table 5.1 should seem natural—with one caveat which we shall address shortly.

§5.3. Inductive Types and their Constructors Hovering above terms and all else we have the types. The extension of the type system with inductive types is as before: given a typescheme $F(X)$, its inductive closure is represented by $\mu X. F(X)$. The Mendler inductive
Types:

\[ T := \ldots \mid \mu X.F(X) \mid \nu X.F(X) \]

Terms:

\[ t := \ldots \mid \text{min}(t) \mid \text{mcoitr}_{r,x}(t,u) \quad k := \ldots \mid \text{mitr}_{\rho,a}[k,l] \mid \text{mout}[k] \]

Typing Rules:

\[ \Gamma \vdash t : F(\mu X.F(X)) \mid \Delta \]

\[ \Gamma \vdash \text{min}(t) : \mu X.F(X) \mid \Delta \]

\[ \Gamma \mid k : F(X) \vdash \Delta, \rho : X \to A, a : A \quad \Gamma \mid l : A \vdash \Delta \]

\[ \Gamma \mid \text{mitr}_{\rho,a}[k,l] : \mu X.F(X) \vdash \Delta \]

\[ X \text{ not free in } \Gamma, \Delta, A \]

\[ x : A, r : A \to X, \Gamma \vdash t : F(X) \mid \Delta \]

\[ \Gamma \vdash u : A \mid \Delta \]

\[ X \text{ not free in } \Gamma, \Delta, A \]

\[ \Gamma \vdash \text{mcoitr}_{r,x}(t,u) : \nu X.F(X) \mid \Delta \]

\[ \Gamma \mid k : F(\nu X.F(X)) \vdash \Delta \]

\[ \Gamma \mid \text{mout}[k] : \nu X.F(X) \vdash \Delta \]

Reduction Rules:

\[ \text{min}(t) \cdot \text{mitr}_{\rho,a}[k,l] \leadsto t \cdot k[\rho.(\text{mitr}_{\rho,a}[k,\beta])][l/a] \]

\[ \text{mcoitr}_{r,x}(t,u) \cdot \text{mout}[k] \leadsto t[\lambda x.(\text{mcoitr}_{r,x}(t,x))][u/x] \cdot k \]

Table 5.1: The types and syntax of DC with Mendler induction
type constructor is an operator parametrized tacitly by a term of type \( F(\mu X.F(X)) \), and for each such combination it yields a term \( \text{min}(t) \) of type \( \mu X.F(X) \):

\[
\Gamma \vdash t : F(\mu X.F(X)) | \Delta \\
\Gamma \vdash \text{min}(t) : \mu X.F(X) | \Delta
\]

§5.4. Eliminator  To understand the workings of the eliminator, let us look at the intuitionistic version:

\[
mitr^{F(X)} A f t.
\]

Armed with an inductive step—which we have determined will be parameters in the classical version—the eliminator consumes a term \( t \) of inductive type; hence, it makes sense to make it a co-term in the classical setting. Developing one step of reduction yields:

\[
f \left( \varepsilon \mu X.F(X). \langle t, \lambda x : \mu X.F(X). mitr^{F(X)} A f x \rangle \right).
\]

The inductive step is such, then, that it also continues from the value of the Mendler restriction (§4.9) created by the inductive step—and will, thus, be also modelled as a co-term. The inductive operation is also expected to return a value; in the classical world, returning can be conflated with passing the return value to the continuation—another parameter we need to take into account. The form of the Mendler induction operator, for an inductive step \( k \) and a continuation \( l \) is

\[
mitr_{\rho,a}[k,l]
\]

The two extra parameters \( a \) and \( \rho \), co-variables each, are specific to the induction step. The induction step receives two arguments and returns some value. One of the arguments is a value having the unrolling of the inductive type as its own type; this can be passed to the inductive step via a cut. The other argument, the induction, is also a continuation; again, we recur to the slogan that to return a value is the same as applying a continuation to it. This can be passed to the inductive step directly via substitution; the two co-variables represent, in the co-term, those points where we want each continuation to be called—\( \rho \) the induction, \( a \) the return continuation. The inductive operator therefore binds \( \rho \) and \( a \) in \( k \) but not in \( l \).

§5.5. Typing the Eliminator  To type the eliminator we need to consider two issues further. First, the induction is not a plain continuation; the continuation step calls the induction, possibly repeatedly, to return the values it needs to compute its own result. This returning (which, we have seen, is actually calling a continuation) is something that continuations cannot in general do. Hence, what we pass to the inductive step is not the induction (a continuation) but its subtractive abstraction

\[
\mu a.(mitr_{\rho,a}[k,a]) : \mu X.F(X) - A.
\]

We use subtraction for it is more natural in this context: we are abstracting on a continuation, not on a value as it happens with (the more familiar) implicative types. Indeed, to represent the induction as a lambda abstraction would require an extra-level of indirection—viz. a co-variable abstraction:

\[
\lambda x.(a.(x * mitr_{\rho,a}[k,a])) : \mu X.F(X) \to A.
\]
The second issue has to do with the other type constructor we find in Mendler induction—the existential quantification. The inductive type must be hidden from the induction step whenever it calls the induction (represented by co-variable \( \rho \)). We will do that by doing what we did in the typing of existential quantifications themselves: by proving the typing with the aid of a stand-in type variable that is not free in the typing contexts.

\[
\begin{align*}
\Gamma | k : F(X) \vdash \Delta, \rho : X \to A, \alpha : A & \quad \Gamma | l : A \vdash \Delta \\
& \quad \Gamma | \text{mitr}_{\rho, \alpha}[k, l] : \mu X.F(X) \vdash \Delta \\
\end{align*}
\]

\((X \text{ not free in } \Gamma, \Delta, A)\)

§5.6. Reduction  The reduction rule follows simply from the aforementioned considerations. When cutting an inductive constructor with an eliminator, we unroll the inductive term, and pass it to the inductive step after this has been suitably instantiated—via substitution—with the inductor and the return continuation.

\[
\text{min}(t) \bullet \text{mitr}_{\rho, \alpha}[k, l] \leadsto t \bullet k[\mu \beta.(\text{mitr}_{\rho, \alpha}[k, \beta]) / \rho][l / \alpha] \quad (\beta \text{ fresh})
\]

§5.7. Co-induction  Co-induction is obtained by direct dualization of the operators introduced above and of their behaviour. Given a typescheme \( F(X) \) its co-inductive closure is represented by \( \nu X.F(X) \). The eliminator \( \text{mout}[k] \) takes a continuation \( k \) of type \( F(\nu X.F(X)) \) to create a continuation of the co-inductive type. The introduction (the co-iterator), \( \text{mcoitr}_{r, x}(t, u) \) takes two arguments: \( t \), the co-induction step, and \( u \), the seed—a piece of data which acts as input to the co-iteration step and thereby determines the flow of the co-induction. The co-iterator also sports two parameter variables: \( x \) is where the seed of the co-induction will be fed into the co-inductive step; \( r \) is where the co-induction step is to be plugged in to obtain, intuitively, the next element of the sequence. The co-inductor is a value—to compose co-inductive calls, it is therefore natural to pass the co-inductive invocation as a lambda abstraction to the co-inductive step—dual to what we did for induction where subtractive abstractions were used. The whole reduction rule is simply the dual of the one defined for Mendler induction.

5.2 Classical Natural Mendler

§5.8. Making it Natural  This complicated flow of information will hopefully become clearer—viz. all the massaging the sub-continuations have to go through—when we look at concrete applications of Mendler induction. The natural choice is the natural numbers; let’s see how to add them (pun intended). Cuts shall reduce via the abstraction prioritizing strategy.

§5.9. Base Types and Terms  We posit a distinguished type variable \( B \), and from this construct the type

\[ 1 \equiv B \lor \neg B \]
inhabited by the familiar

\[
\star \equiv a.(t_2(\text{not}(x.(t_1(x) \cdot a))) \cdot a).
\]

The base “functor” for the naturals is the usual \(F(X) \equiv 1 \lor X\). The Mendler naturals will be those terms of type \(\mathcal{N} = \mu X.F(X)\). We can define the “numbers” by

\[
\begin{align*}
\text{zero} & \equiv \text{min}(t_1(\star)) \\
\text{one} & \equiv \text{min}(t_2(\text{zero})) \\
\text{two} & \equiv \text{min}(t_2(\text{one}))
\end{align*}
\]

The successor “function” cannot be a function without functional types. However, we can construe it instead as a continuation that takes some natural \(x\), adds the outer injections (into the disjunction and the inductive type), and then continues with some given continuation \(k\):

\[
succ^k = x.(\text{min}(t_2(x)) \cdot k) \quad (x \notin fv(k)).
\]

§5.10. Addition Mendler-inductively As in the usual addition of natural numbers, the algorithm distinguishes two cases. If the first argument is zero, then we “return” the second argument—call it \(m\)—but returning in Mendler induction is to pass the result to the output co-variable (here \(\alpha\)):

\[
x.(m \cdot \alpha) \quad (x, \alpha \notin fv(m)).
\]

If the first argument is not zero, we compute the inductive invocation and then apply the successor to it. Due to the use of continuations, this is done by passing both the value we want to call the inductor on and the successor—the two neatly packed in a subtractive value—to the inductor represented by the co-variable \(\rho\):

\[
x.(x# \text{succ}^n) \cdot \rho).
\]

Putting it all together, for any “natural” term \(m\), our inductive step is

\[
\text{Step}^m \equiv [x.(m \cdot \alpha), x.(x# \text{succ}^n) \cdot \rho]).
\]

For any two “natural” terms \(n\) and \(m\), we define the computation of their sum by

\[
n \cdot \text{mitr}_{\rho,\alpha}[\text{Step}^m, l].
\]

§5.11. Correctness We claim this definition is correct under the abstraction prioritizing reduction strategy in that it calls the continuation \(l\) with the “natural” corresponding to the sum of the naturals represented by \(n\) and \(m\). We can see this for simple examples by calculating:

\[
\begin{align*}
\text{zero} \cdot \text{mitr}_{\rho,\alpha}[\text{Step}^m, l] & \rightarrow t_1(\star) \cdot \text{Step}^m[\mu a.(\text{mitr}_{\rho,\alpha}[\text{Step}^m, \alpha]) / \rho][l / \alpha] \\
& \rightarrow \star \cdot x.(m \cdot l) \\
& \rightarrow m \cdot l;
\end{align*}
\]
and

$$\text{one} \cdot \text{mitr}_{\rho,a}[\text{Step}^n, l] \rightarrow t_2(\text{zero}) \cdot \text{Step}^m[\mu a.(\text{mitr}_{\rho,a}[\text{Step}^m, a]) / \rho][l/\alpha]$$

$$\rightarrow \text{zero} \cdot x.(x \# \text{succk}^l) \cdot \mu a.(\text{mitr}_{\rho,a}[\text{Step}^m, a])$$

$$\rightarrow \left(\text{zero} \# \text{succk}^l\right) \cdot \mu a.(\text{mitr}_{\rho,a}[\text{Step}^m, a])$$

$$\rightarrow \text{zero} \cdot \text{mitr}_{\rho,a}[\text{Step}^m, \text{succk}^l]$$

$$\rightarrow m \cdot \text{succk}^l$$

$$\rightarrow \min(t_2(m)) \cdot l.$$

We can turn this into a more general, inductive argument. By abuse of notation, we term the sum in this representation of two actual natural numbers $n$ and $m$ by $n + m$. We can then reason

$$\min(t_2(n)) \cdot \text{mitr}_{\rho,a}[\text{Step}^m, l] \rightarrow t_2(n) \cdot \text{Step}^m[\mu a.(\text{mitr}_{\rho,a}[\text{Step}^m, a]) / \rho][l/\alpha]$$

$$\rightarrow n \cdot x.(x \# \text{succk}^l) \cdot \mu a.(\text{mitr}_{\rho,a}[\text{Step}^m, a])$$

$$\rightarrow \left(n \# \text{succk}^l\right) \cdot \mu a.(\text{mitr}_{\rho,a}[\text{Step}^m, a])$$

$$\rightarrow n \cdot \text{mitr}_{\rho,a}[\text{Step}^m, \text{succk}^l]$$

$$\rightarrow n \cdot \text{succk}^l$$

$$\rightarrow \min(t_2(n + m)) \cdot l.$$ (IH)

Note how in eliminating the input value, we accumulate the rest of the computation in the continuation argument of the iterator.

### 5.3 Classical Mendler Subsuming Intuitionistic

#### §5.12. Translation

We have claimed that DC with Mendler induction is a powerful programming system. Our journey towards backing this claim up begins with the encoding of terms of the functional flavour of Mendler induction presented in chapter 4 as terms of DC (table 5.2). This translation is somewhat peculiar in that it uses subtraction to represent lambda-abstractions. There is no binding reason for this—it increases the complexity of the phrases in the target of the translation—but we wish to stress that it is possible to do so. Another notable feature of this translation is that it removes any and all type annotations in the terms. Further, we can remove universal types altogether from the target of the translation since they are not required by the Classical Mendler inductor.\footnote{As is well-known, the type annotations of System F are crucial in establishing that terms are well-typed and thereby strongly normalizing, but they play no significant part in reduction and can be safely erased before performing it (Gallier, 1989).}
\[ \overline{X} = X \]
\[ A \land B = \overline{A} \land \overline{B} \]
\[ A \to B = \neg(\overline{A} \to \overline{B}) \]
\[ \forall X. F(\overline{X}) = F(X) \]
\[ \exists X. F(\overline{X}) = \exists X. F(X) \]
\[ \mu X. F(\overline{X}) = \mu X. F(X) \]

\[
\begin{align*}
\overline{x} &= x \\
\langle t, t' \rangle &= \langle \overline{t}, \overline{t}' \rangle \\
\lambda x : T. t &= \text{not}(\mu \alpha. (x \cdot (\overline{t} \cdot \alpha))) \\
\overline{\lambda x : T. t} &= \overline{\lambda x : T. t} = i \\
\overline{\text{let } X, x = t \text{ in } u} &= \alpha. (\overline{\overline{t}} \cdot e[x \cdot (\overline{u} \cdot \alpha)]) \\
\overline{\text{min}^f(X)} &= \text{not}(\mu \alpha. (x \cdot (\text{min}(x) \cdot \alpha))) \\
\overline{\text{mitr}^f(X)} &= \text{not}(\mu \alpha. (\text{let } X, x = t \text{ in } u = \alpha. (\overline{\overline{t}} \cdot e[x \cdot (\overline{u} \cdot \alpha)])))
\end{align*}
\]

Table 5.2: Translation of Intuitionistic Mendler induction into the Dual Calculus with Mendler induction.
Theorem 18 (Soundness). The translation of table 5.2 preserves typings: for any term \( t \), typing context \( \Gamma \) and type \( T \)

\[
\Gamma \vdash t : T \implies \overline{\Gamma} \vdash \overline{t} : \overline{T}
\]

Proof. Induction on the typing derivations. \( \square \)

§5.13. Simulating Conversions  The theory of System F of the previous chapter was based on conversion. This means that the relation is immediately reflexive, symmetric, transitive, and congruent (conversions can happen inside any sub-term). The theory of DC with Mendler induction is given by reduction rules—a more restrictive notion. In comparing the two formulations we must relax slightly our requirements: Instead of proving that the translation of a term reduces to the translation of another, we prove merely that they reduce to a common phrase; we also accept that reduction may occur inside a term to handle congruent conversions, and accept the extensionality principle for co-abstractions, wherein for fresh \( \alpha, \alpha.(t \cdot \alpha) = t \).

Remark 3. The translation satisfies the usual term and type substitution lemmas, and, additionally, for any term \( t \), type variable \( X \), and type \( T \)

\[
\overline{t} = t[T/X].
\]

We refrain from proving these facts here—straightforward inductions suffice.

Theorem 19. Let \( t =_\beta u \) be terms of System F with Mendler induction. If we take the congruent closure of the abstraction-first reduction discipline of DC we have, for any co-variable \( \alpha \), that there is a term \( v \) such that

\[
\overline{t} \cdot \alpha \rightsquigarrow^* v \cdot \alpha = \overline{u} \cdot \alpha
\]

Proof. We argue by case analysis on the conversion relation. Most cases are simple; we focus on the conversion rule for induction.

\[
\text{mitr}^{F(X)} A f \left( \text{min}^{F(X)} t \right) \cdot \alpha
\]

\[
= \alpha. \left( a. \left( \text{mitr}^{F(X)} \cdot \text{not} \left( \left( \overline{f} \cdot \alpha \right) \right) \right) \cdot \text{not} \left( \left( \text{min}^{F(X)} t \cdot \alpha \right) \right) \right) \cdot \alpha
\]

\[
\rightsquigarrow \alpha. \left( \text{mitr}^{F(X)} \cdot \text{not} \left( \left( \overline{f} \cdot \alpha \right) \right) \right) \cdot \text{not} \left( \left( \text{min}^{F(X)} t \cdot \alpha \right) \right)
\]

\[
\Rightarrow \text{mitr}^{F(X)} \cdot \text{not} \left( \left( \overline{f} \cdot \text{not} \left( \left( \text{min}^{F(X)} t \cdot \alpha \right) \right) \right) \right)
\]

\[
= \text{not} \left( \mu a. \left( f. \left( \text{not} \left( \mu a. \left( \text{mitr}_{p,\beta} \left[ z. \left( f \cdot \text{not} \left( \left( e \left( \langle z, \text{not} (\rho) \rangle \# \beta \right) \right) , \alpha \right) \right) \right) \right) \right) \right) \right) \cdot \alpha \right) \right)
\]

\[
\Rightarrow \text{not} \left( \mu a. \left( \text{mitr}_{p,\beta} \left[ z. \left( \overline{f} \cdot \text{not} \left( \left( e \left( \langle z, \text{not} (\rho) \rangle \# \beta \right) \right) , \alpha \right) \right) \right) \right) \cdot \alpha \right) \right)
\]

\[
\Rightarrow \left( \text{min}^{F(X)} t \cdot \alpha \right) \cdot \mu a. \left( \text{mitr}_{p,\beta} \left[ z. \left( \overline{f} \cdot \text{not} \left( \left( e \left( \langle z, \text{not} (\rho) \rangle \# \beta \right) \right) , \alpha \right) \right) \right) \right)
\]
The other side of the equation simplifies to

\[
\langle t, (\min^* \mu \cdot \text{itr}_{\rho,\beta} \cdot [z. (\mu \cdot \text{not} (e \langle z, \text{not}(\rho) \rangle \# \beta)) \cdot \alpha] \rangle \cdot \alpha \\
\cdot (\mu \cdot \text{not} (e \langle z, \text{not}(\rho) \rangle \# \beta)) = \gamma) / \rho [\alpha / \beta] \\
\cdot (\mu \cdot \text{not} (e \langle z, \text{not}(\rho) \rangle \# \beta)) \cdot \alpha) \\
= \langle t, z. (\mu \cdot \text{not} (e \langle z, \text{not}(\rho) \rangle \# \beta)) \rangle \cdot \alpha) \\
\cdot (\mu \cdot \text{not} (e \langle z, \text{not}(\rho) \rangle \# \beta)) \cdot \alpha) \\
\cdot (\mu \cdot \text{not} (e \langle z, \text{not}(\rho) \rangle \# \beta)) \cdot \alpha)
\]

The two sides develop to outwardly similar cuts; they distinguish each other in sub-cuts which, we will show are intra-convertible. The cut marked with † contains the sub-cut

\[
\langle t, z. (\mu \cdot \text{not} (e \langle z, \text{not}(\rho) \rangle \# \beta)) \rangle \cdot \alpha \\
\cdot (\mu \cdot \text{not} (e \langle z, \text{not}(\rho) \rangle \# \beta)) \cdot \alpha)
\]

Therefore, assuming congruence and \( \eta \)-equality, we can develop the cut † to

\[
\langle t, z. (\mu \cdot \text{not} (e \langle z, \text{not}(\rho) \rangle \# \beta)) \rangle \cdot \alpha \\
\cdot (\mu \cdot \text{not} (e \langle z, \text{not}(\rho) \rangle \# \beta)) \cdot \alpha)
\]

just as above.
§5.4 Classical Induction

Like in the previous chapter, we will now compare Mendler with ordinary induction in DC. Kimura and Tatsuta (2009) proposed one such system that shall be the basis of our analysis. In fact this system—which they termed $DC_{\mu\nu}$—also included co-induction; here, however we shall only be interested in the inductive component. They proved that $DC_{\mu\nu}$ was strongly normalising—ultimately via an embedding into the symmetric lambda calculus of Barbanera and Berardi (1996). Having understood Mendler induction as laid out in these pages, this system will not feel unfamiliar and so we will skim through the similarities and focus on the differences.

§5.15. The Extension

The inductive extension of Kimura and Tatsuta (2009) is based on the purely propositional version of DC, including only conjunction, disjunction and negation. To this it adds inductive types and the associated inductive operators, together with their typing and reduction rules (table 5.3). At the level of operators, we need to add an inductive type constructor which for every typescheme $F(X)$ generates the type $\mu X. F(X)$.

On the syntactical level, the constructor for $\mu X. F(X)$ is an operator on terms $t$ of type $F(\mu X. F(X))$, satisfying the typing rule

$$
\Gamma \vdash t : F(\mu X. F(X)) \mid \Delta \\
\Gamma \vdash \text{in}^{F(X)}(t) : \mu X. F(X) \mid \Delta
$$

On the co-term side where the induction lives, we find a few differences. The first is that, like in the Intuitionistic case, ordinary induction is characterised by managing the inductive calls and passing the results to the inductive step; the induction co-variable is thus unnecessary. This also explains why, here, we do not need to add functional types. The return co-variable, however, remains, as do the two parameters to the operator, the inductive step, and the continuation. The eliminator and its typing rule are then

$$
\Gamma \mid k : F(A) \vdash \Delta, \alpha : A \quad \Gamma \mid l : A \vdash \Delta \\
\Gamma \mid \text{itr}^{F(X)}_{A,\alpha}[k, l] : \mu X. F(X) \vdash \Delta
$$

§5.16. Maps

To understand the way in which induction reduces let us look at the intuitionistic case

$$
\text{itr}^{F(X)} A f \left(\text{in}^{F(X)} t\right) = f \left(F \mu F A \left(\text{itr}^{F(X)} A f\right) t\right).
$$

What it all boils down to is that the iterator unwraps the term $t$ inside inductive constructor, then applies the map of the iteration to $t$ and passes the result to the iteration step $f$. It is clear, then, that before going any further we must find a way to define this map. We

\footnote{This is not a coincidence. Prior to attacking the problem of Mendler Induction in the Dual Calculus, the author studied this system in detail and, naturally, it influenced later developments.}
CLASSICAL INDUCTION

Types:

\[ T := \ldots | \mu X . F(X) \]

Terms:

\[ t := \ldots | in^{F(X)}(t) \]
\[ k := \ldots | itr_{A,a}^{F(X)}[k,k'] \]

Typing Rules:

\[
\Gamma \vdash t : F(\mu X . F(X)) \mid \Delta \\
\Gamma \vdash in^{F(X)}(t) : \mu X . F(X) \mid \Delta \\
\Gamma \vdash \mu X : F \mid \Delta \\
\Gamma \vdash \mu X : F, X/\Delta \\
\Gamma \vdash(itr_{A,a})^{F(X)}[k,l] : \mu X . F(X) \mid \Delta 
\]

Reduction Rules:

\[ in^{F(X)}(t) \cdot itr_{A,a}^{F(X)}[k,l] \rightarrow t \cdot kmap^{F(X)}_{\mu X . F(X), A, \beta}[itr_{\mu X . F(X), a}^{F(X)}[k, \beta], k[l/\alpha]] \]

\[ kmap_{A_B, \beta}^{X(X)}[k, l] = x.(\beta \cdot (x \cdot k) \cdot l) \]
\[ kmap_{\beta}^{T(X)}[k, l] = l \quad (X \not\in ftv(T)) \]
\[ kmap_{A_B, \beta}^{F(X) F(X)}[k, l] = \pi \left( \gamma, \left( p \cdot \text{fst}\left[kmap_{A_B, \beta}^{F(X)}[k, \gamma]\right]\right), \gamma, \left(p \cdot \text{snd}\left[kmap_{A_B, \beta}^{F(X)}[k, \gamma]\right]\right) \right) \cdot l \]
\[ kmap_{A_B, \beta}^{F(X) F(X)}[k, l] = \left[ kmap_{A_B, \beta}^{F(X)}[k, \text{z} \cdot (t_1(z) \cdot l)], kmap_{A_B, \beta}^{F(X)}[k, \text{z} \cdot (t_2(z) \cdot l)] \right] \]
\[ kmap_{A_B, \beta}^{F(X) F(X)}[k, l] = \text{not}\left[ \gamma, \left( \text{not}\left( kmap_{A_B, \beta}^{F(X)}[k, \gamma]\right) \right), l \right] \]
\[ kmap_{A_B, \beta}^{F(X) F(X)}[k, l] = \text{itr}_{\mu Y . F(B,Y)/Y}^{F(X, \mu Y . F(B,Y))}[kmap_{A_B, \beta}^{F(X, \mu Y . F(B,Y))}[k, x \cdot (\text{in}^{F(B,Y)}(x) \cdot \gamma)], l] \]

\[ (Y \not\in ftv(A, B); \gamma, z \text{ fresh}) \]

Table 5.3: Extension of the propositional DC with inductive types
cannot simply mimic the intuitionistic case for we do not have functional types. One way of doing this is by utilizing a mapping function which is external to DC—i.e. something which given a continuation (for the induction is a continuation) will return a co-term where the induction is applied in a type suitable way and which then calls the induction step.

More formally, we define (in table 5.3) a function

$$kmap^{F(X)}_{A,B,\beta}[\cdot,\cdot] : \mathcal{K} \times \mathcal{K} \to \mathcal{K};$$

by induction on typescheme $F(X)$. The first argument is the continuation of type $A$ we wish to map; the (meta) co-variable $\beta$ represents its output, of type $B$, which the map must capture—i.e. it will not appear free in the resulting co-term—in order to compose the different invocations of the mapped co-term when, for example, we encounter a pair. The second argument is the continuation which will proceed with the value of type $F(B)$ constructed by the map invocation. For example, when we’re mapping $X$ in the typescheme $X$, $kmap^X_{A,B,\beta}[k,l]$, all we have to do is take the value (type $A$), apply $k$ and catch the output $\beta$ (type $B$) and pass it to $l$,

$$x.(\beta(x \cdot k) \cdot l).$$

On the other hand, when the map is on a type $T$ which does not include the typescheme type variable $X$ free—and therefore there is no actually mapping to be done—we simply continue with the output continuation $l$.

The conjunctive and disjunctive cases follow naturally—they just apply the map inductively on the subtypes and use their constructors to build something of the respective type to be passed to $l$. The negation case is not dissimilar; however, as a testimony of the contra-variance of negation, the input and output types swap. Finally, the inductive type requires its own instance of the mapping function. The definition is more elaborate, if anything because the definition now involves two type variables: one for the mapping, and the other for the induction. Nonetheless, the pattern is the same: apply the eliminator to access the sub-types; apply inductively the mapping function; and continue with the output continuation.

§5.17. Soundness of Mapping This definition is type-sound in the sense that it takes well-typed co-terms to well-typed co-terms (under the proviso that the parametrizing typeschemes are co-variant).

**Theorem 20** (Map is Type-sound). Fix a type $F(X)$, where $X$ does not appear negatively. For any $k$ and $l$, $\beta$, and $A$ and $B$ and typing (co-) contexts $\Gamma$ and $\Delta$ such that

$$\Gamma \mid k : A \vdash \Delta, \beta : B \quad \text{and} \quad \Gamma \mid l : F(B) \vdash \Delta$$

it is the case that

$$\Gamma \mid kmap^{F(X)}_{A,B,\beta}[k,l] : F(A) \vdash \Delta.$$

Otherwise, should $X$ not appear positively in $F(X)$, and under the same conditions as before except that

$$\Gamma \mid l : F(A) \vdash \Delta,$$

it is the case that

$$\Gamma \mid kmap^{F(X)}_{B,A,\beta}[k,l] : F(B) \vdash \Delta.$$
Proof. Proof is by mutual induction, as forced by the contra-variance of negation.

§5.18. Reduction Returning to reduction, endowed with the ability to map co-terms, the reduction reduces to unwrapping the term of inductive type, mapping itself with a new output continuation to match the output co-variable of the map, and continuing with the inductive step—whose output is now the overall continuation $l$

\[ \text{in}_{F(X)}(t) \cdot \text{itr}_{A,a}^{F(X)}[k, l] \leadsto t \cdot \text{kmap}_{\mu X.F(X),a,\beta}^{F(X)} \left[ \text{itr}_{\mu X.F(X),a}^{F(X)}[k, \beta], k[l/a] \right]. \]

5.5 Classical Mendler Induction Subsumes Classical Induction

§5.19. Embedding into Mendler We will now show that there is a natural way of translating inductive programs into our system of Mendler induction. The exact source of the translation is the inductive variant of DC from the previous section (cf. table 5.4). Most cases are trivial following from homomorphic application of the translation to sub-phrases. The exception is (obviously) the iteration case. Here we need to explicitly apply the induction using a functorial action $\text{kmmap}_{\beta}^{F(X)}[k, l]$, as was done in the functional setting of section 4.3:

\[ \text{itr}_{A,a}^{F(X)}[k, l] = \text{mitr}_{\rho,\alpha}^{\mu X.F(X),a,\beta} \left[ \text{kmmap}_{\beta}^{F(X)}[x.(x \# \beta) \cdot \rho], k[l/a] \right]. \]

In the first argument continuation of the functorial action, co-variable $\rho$ stands for the induction—which it will be replaced with during the reduction of the Mendler iterator—and applies to it the argument on which the induction is to be performed together with the (captured) continuation $\beta$ of the map. The functorial action then passes the result to $k$, the iteration step.

§5.20. Mendler Mapping The central issue that we are left with, then, is the existence or not of the aforementioned mapping co-terms, suitable enough to apply the induction to the inductive sub-terms of an inductive term. For most composite types the situation is simple enough (cf. table 5.5). As we have seen in the previous section, the general recipe is: get the term of composite type; use the eliminator to access the components of the sub-types; apply the map for that type; and use the constructor to get back to the composite type. This recipe applies to all types; however, seeing how this works for inductive types puts us in a slightly more delicate position because we must—as is done in the ordinary map—perform another iteration on a different type variable; and therefore we need to perform an additional mapping to simulate this induction.

In the sequel, we will see our typeschemes as being parametrized on a vector of type-variables. The definition of the mapping co-terms is done by induction on the structure of these typeschemes; in particular, a type-scheme $F(\vec{X}, Y)$ is considered smaller than the corresponding inductive closure $\mu Y.F(\vec{X}, Y)$—even though the latter has one fewer parameter. The induction step must accept something of type $F(\vec{A}, \mu X.F(\vec{B}, X))$—where $\vec{A}$ and $\vec{B}$ are vectors of types that instantiate the parameters of the type-scheme other
\[
\bar{T} = T
\]
\[
\bar{x} = x
\]
\[
\langle t, t' \rangle = \langle \bar{t}, \bar{t}' \rangle
\]
\[
\bar{\alpha} = \alpha
\]
\[
\bar{\text{fst}[k]} = \text{fst}[\bar{k}]
\]
\[
\bar{\text{snd}[k]} = \text{snd}[\bar{k}]
\]
\[
\bar{\text{not}[k]} = \text{not}[\bar{k}]
\]
\[
\bar{\text{min}^{F(X)}[t]} = \text{min}^{F(X)}[\bar{t}]
\]
\[
\bar{\text{mitr}_{\rho, a}^{F(X)}[k, l]} = \text{mitr}_{\rho, a}^{F(X)}[\bar{k}, \bar{l}]
\]
\[
\bar{\alpha.(c)} = \alpha.(\bar{c})
\]
\[
\bar{t \cdot k} = \bar{t} \cdot \bar{k}
\]

Table 5.4: Translation of Classical induction into Classical Mendler Induction

than the induction type variable, which takes the inductive type for its value. These extra parameters make an appearance at the level of the continuations that are mapped: we assume that for a list of parameters \( \vec{X} \) of length \( n \), each \( k_i \) is a continuation of type \( A_i \), which outputs for some co-variable \( \beta_i (0 < i \leq n) \). These \( \beta_i \) are assumed to be distinct and also assumed not to appear free anywhere outside of their respective continuation \( k_i \).

§5.21. Soundness of Translation

The translation of the system with classical induction into the system with Mendler induction is sound in the sense that it takes well-typed terms to well-typed terms. Because of the dependence of the translation on mapping functions, we also need to show that the mapping is sound—that it behaves on types like a functor, extending a continuation on \( F(\vec{B}) \) to one \( F(\vec{A}) \) via the actions of the continuations \( k_i : A_i \), which output each on co-variable \( \beta_i : B_i \). Additionally, we can prove that the map construction is sensibly defined in that it commutes with the translation: the translation of the (ordinary) mapping function equals the Mendler mapping of the translated arguments.

**Theorem 21.** (Map is well-defined) Fix a typescheme \( F(\vec{X}) \), where all \( \vec{X} \) do not appear negatively. For any \( k \) and \( l \), \( \vec{\beta} \), and \( A \) and \( B \) and typing (co-) contexts \( \Gamma \) and \( \Delta \) such that

\[
\Gamma \mid k_i : A_i \vdash \Delta, \beta_i : B \quad \text{and} \quad \Gamma \mid l : F(\vec{B}) \vdash \Delta
\]

it is the case that

\[
\Gamma \mid \text{kmmap}^{F(\vec{X})}_\beta[\bar{k}, \bar{l}] : F(\vec{A}) \vdash \Delta.
\]
\[ k\text{mmap}_{\vec{\beta}}^{X(\vec{\alpha})} \left[ \vec{k}, l \right] = x_{\vec{i}} \cdot (x \cdot l) \]
\[ k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, l \right] = l \quad \text{(}X_1, \ldots, X_n \notin \text{ftv}(T)\text{)} \]
\[ k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, l \right] = p_{\vec{i}} \cdot (\gamma_{\vec{i}} \cdot (p \cdot \text{fst} \left[ k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, \vec{\gamma} \right] \right]) \cdot (p \cdot \text{snd} \left[ k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, \vec{\gamma} \right] \right])) \cdot l \]
\[ k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, l \right] = k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, z_{\vec{i}} (z \cdot l) \right] \]
\[ k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, l \right] = \text{not} \left[ \gamma_{\vec{i}} \cdot (\text{not} \left[ k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, \vec{\gamma} \right] \right]) \cdot l \right] \]
\[ k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, l \right] = \text{mitr}_{\vec{\rho}, \vec{\alpha}} \left[ k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{\beta}, z ((z \# \vec{\beta}) \cdot \vec{\rho}) , k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, \vec{\beta}, x \cdot (\text{min} (x) \cdot \vec{\alpha}) \right] \right], l \right] \]

Table 5.5: The Mendler version of the mapping for continuations

Otherwise, should the \( \vec{X} \) not appear positively in \( F(\vec{X}) \), and under the same conditions as before except that for

\[ \Gamma | l : F(\vec{A}) \vdash \Delta \]

it is the case that

\[ \Gamma | k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, l \right] : F(\vec{B}) \vdash \Delta. \]

**Proof.** Proof is by mutual induction on the parameter typescheme, as forced by the contra-variance of negation. There is only one truly interesting case.

**Inductive Types:** Our end goal is to prove that

\[ \Gamma | \text{mitr}_{\vec{\rho}, \vec{\alpha}} \left[ k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{\beta}, z ((z \# \vec{\beta}) \cdot \vec{\rho}) , k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, \vec{\beta}, x \cdot (\text{min} (x) \cdot \vec{\alpha}) \right] \right], l \right] : \mu Y.F(\vec{A}, Y) \vdash \Delta \]

Since \( \Gamma | l : \mu Y.F(\vec{B}, Y) \vdash \Delta \), the typing rule for the Mendler iterator requires

\[ \Gamma | k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{\beta}, z ((z \# \vec{\beta}) \cdot \vec{\rho}) , k\text{mmap}_{\vec{\beta}}^{F(\vec{\alpha})} \left[ \vec{k}, \vec{\beta}, x \cdot (\text{min} (x) \cdot \vec{\alpha}) \right] \right] : F(\vec{A}, Y) \vdash \Delta, \rho : Y - \mu Y.F(\vec{B}, Y), \vec{\alpha} : \mu Y.F(\vec{B}, Y) \]

We can use the induction hypothesis (on the smaller typescheme \( F(\vec{X}, Y) \)) because

\[
\begin{align*}
z : Y \vdash (z \# \vec{\beta}) : Y - \mu Y.F(\vec{B}, Y) & \vdash \beta : \mu Y.F(\vec{B}, Y) \\
| \beta_i : A_i \vdash \beta_i : A_i, & \quad z : Y \vdash (z \# \vec{\beta}) : \rho \vdash Y - \mu Y.F(\vec{B}, Y), \beta : \mu Y.F(\vec{B}, Y) \\
& \quad | z ((z \# \vec{\beta}) : \rho) : Y \vdash \rho : Y - \mu Y.F(\vec{B}, Y), \beta : \mu Y.F(\vec{B}, Y)
\end{align*}
\]

131
if we can also prove that

\[ \Gamma | \text{kmmap}^{F(X,Y)}_\beta \cdot \alpha : F(\tilde{X}, \mu Y.F(\tilde{B},Y)) \vdash \Delta, \alpha : \mu Y.F(\tilde{B},Y) \]

This we can achieve by a second use of the induction hypothesis since:

\[ \Gamma | k_i : A_i \vdash \Delta, \beta : B_i | \beta : \mu Y.F(\tilde{B},Y) \]

and

\[ x : F(\tilde{B}, \mu Y.F(\tilde{B},Y)) \vdash x : F(\tilde{B}, \mu Y.F(\tilde{B},Y)) : \mu Y.F(\tilde{B},Y) | \Delta, \]

\[ x : F(\tilde{B}, \mu Y.F(\tilde{B},Y)) \vdash \text{min}(x) : \mu Y.F(\tilde{B},Y) | \Delta, \]

\[ | x.(\text{min}(x) \cdot \alpha) : F(\tilde{B}, \mu Y.F(\tilde{B},Y)) \vdash \Delta, \alpha : \mu Y.F(\tilde{B},Y) \]

The negative case is similar.

\[ \square \]

\textbf{Theorem 22 (Type Preservation).} Typings in DC with inductive types, induce via the given translation, well-typed terms in the system with Mendler induction:

\[ \Gamma \vdash t : T \mid \Delta \implies \Gamma \vdash \bar{t} : T \mid \Delta \]

\[ \Gamma | k : T \vdash \Delta \implies \Gamma | \bar{k} : T \vdash \Delta \]

\[ \Gamma \vdash c \vdash \Delta \implies \Gamma \vdash \bar{c} \vdash \Delta \]

\textit{Proof.} Type preservation follows from induction as usual, generalising for arbitrary type contexts. However, because the type translation is the identity on types, and the propositional terms and their typings come from DC, most cases are trivial to the extreme.

We shall, then, focus only on inductive types.

The constructor is simple. For

\[ \Gamma \vdash t : F(\mu F) \mid \Delta \]

we have

\[ (\Gamma \vdash \bar{t} : F(\mu F) \vdash \Delta) \text{IH} \]

\[ \Gamma \vdash \bar{t} : F(\mu F) \vdash \Delta \]

\[ \Gamma \vdash \text{min}(\bar{t}) : \mu F \vdash \Delta \]

\[ \Gamma \vdash \text{min}^{F(X)(t)} : \mu F \vdash \Delta \]

For the eliminator, applying the induction hypothesis yields

\[ \Gamma | \bar{k} : F(A) \vdash \Delta, \alpha : A \quad \text{and} \quad \Gamma | \bar{t} : A \vdash \Delta. \]

Now, very easily, we also have that for \( X \notin \text{ftv}(A) \)

\[ | x.(x\beta) \cdot \rho) : X \vdash \beta : A, \rho : X - A \]

132
so that, by theorem 21, we get

\[ \Gamma \mid k\text{mmap}_{\beta}^{F(X)}\left[ x.(x#\beta) \cdot \rho, \overline{k} \right] : F(X) \vdash \Delta, \rho : X \rightarrow A, \alpha : A, \]

and, thereby,

\[ \Gamma \mid m\text{itr}_{\rho,a} \left[ k\text{mmap}_{\beta}^{F(X)} \left[ x.(x#\beta) \cdot \rho, \overline{k} \right], \overline{\tilde{l}} \right] : \mu X.F(X) \vdash \Delta. \]

\[ \Box \]

§5.22. Relating the Two Maps To relate the two maps—the inductive and the Mendler inductive version—we must generalize the inductive mapping of the previous section to the multiple parameter situation. This is done straightforwardly, adapting the definition to the situation where we take vectors of co-variables \( \overline{\beta} \) and continuations \( \overline{\tilde{k}} \). Having levelled the playing field, the relationship between the two is exactly what we would expect: the translation of the inductive mapping is the Mendler mapping of the translation of the parameters.

Theorem 23 (Mapping Relations). Given a typescheme \( F(\overline{X}) \), for any co-terms \( k \) and \( l \),

\[ k\text{map}^{F(\overline{X})}_{\overline{\tilde{k}}, l} = k\text{mmap}^{F(\overline{X})}_{\overline{\beta}} \left[ \overline{k}, \alpha \right] \]

where \( \overline{k} \) is the pointwise application of the translation to each \( k_i \).

Proof. We induct on the typescheme \( F(\overline{X}) \) that parametrize the mapping operator. The case that forces the move to vectors of parameters, the inductive one, goes as follows:

\[ k\text{map}^{\mu Y.F(\overline{X}, Y)}_{\overline{\tilde{k}}, l} = \text{itr}_{\mu Y.F(\overline{B}, Y), a} \left[ k\text{map}^{\mu Y.F(\overline{X}, Y)}_{\overline{\tilde{k}}, l} \left[ \overline{k}, x.\left( \text{in}^F(\overline{B}, Y)(x) \cdot \alpha \right) \right], \overline{\tilde{l}} \right] \]

\[ = m\text{itr}_{\rho,a} \left[ k\text{map}^{\mu Y.F(\overline{X}, Y)}_{\overline{\tilde{k}}, l} \left[ \overline{\beta}, z.(z#\beta) \cdot \rho \right], k\text{map}^{\mu Y.F(\overline{X}, Y)}_{\overline{\tilde{k}}, l} \left[ \overline{k}, \beta, x.\left( \text{in}^F(\overline{B}, Y)(x) \cdot \alpha \right) \right], \overline{\tilde{l}} \right] \]

\[ = m\text{itr}_{\rho,a} \left[ k\text{map}^{\mu Y.F(\overline{X}, Y)}_{\overline{\tilde{k}}, l} \left[ \overline{\beta}, z.(z#\beta) \cdot \rho \right], k\text{map}^{\mu Y.F(\overline{X}, Y)}_{\overline{\tilde{k}}, l} \left[ \overline{k}, \beta, x.(\text{min}(x) \cdot \alpha) \right], \overline{\tilde{l}} \right] \]

\[ = k\text{map}^{\mu Y.F(\overline{X}, Y)}_{\overline{\tilde{k}}, l} \]

\[ \Box \]
§5.23. Substitution

As always, reduction—of abstractions and, also, induction—may proceed with the aid of substitution. In showing that the translation preserves reduction—our ultimate goal—we must show, first, that it is also preserved by substitution. This, in turn, requires us to prove that the mapping on which the iteration translation depends on commutes with substitution (under the proviso that all the variables and co-variables introduced by the translation are fresh for the co-term being subbed-in).

Lemma 19 (Map Substitution). For any type scheme $F(X)$,

$$\text{kmmap}_F^X \left[ k, l \right] [m/\zeta] = \text{kmmap}_F^X \left[ k[m/\zeta], l[m/\zeta] \right]$$

where $k[m/\zeta]$ is the pointwise application of the substitution to each $k_i$.

Proof. Induction on the parametrizing typescheme.

Lemma 20 (Substitution). Term and co-term substitution commute with the translation of Classical induction into Classical Mendler induction:

$$\bar{t}[u/z] = t[u/z] \quad \bar{t}[u/z] = t[m/\zeta]$$

$$k[u/z] = k[u/z] \quad k[u/z] = k[m/\zeta]$$

$$\bar{c}[u/z] = c[u/z] \quad \bar{c}[u/z] = c[m/\zeta]$$

Proof. Mutual induction on the definition of terms, co-terms, and cuts. For the inductive eliminator, the result will follow from the combination of the induction hypothesis, which gives us

$$k[u/z] = k[u/z] \quad \text{and} \quad \bar{t}[u/z] = t[u/z]$$

and the substitutivity property for map (lemma 19), which, given that $x.((x#\beta) \cdot \rho)[u/z] = x.((x#\beta) \cdot \rho)$ tells us that

$$\text{kmmap}_F^X \left[ x.((x#\beta) \cdot \rho), \bar{k}[u/z] \right] = \text{kmmap}_F^X \left[ x.((x#\beta) \cdot \rho), k[u/z] \right];$$

hence,

$$\text{itr}_{A,a}^{F(X)} \left[ k[l], u/z \right] = \text{mitr}_{\rho,a}^{F(X)} \left[ \text{kmmap}_F^X \left[ x.((x#\beta) \cdot \rho), \bar{k}[u/z] \right], \bar{t}[u/z] \right]$$

$$= \text{mitr}_{\rho,a}^{F(X)} \left[ \text{kmmap}_F^X \left[ x.((x#\beta) \cdot \rho), \bar{k}[u/z], \bar{t}[u/z] \right] \right]$$

$$= \text{mitr}_{\rho,a}^{F(X)} \left[ \text{kmmap}_F^X \left[ x.((x#\beta) \cdot \rho), k[u/z], l[u/z] \right] \right]$$

$$= \text{mitr}_{\rho,a}^{F(X)} \left[ k[u/z], l[u/z] \right]$$

$$= \text{itr}_{A,a}^{F(X)} \left[ k[l], u/z \right]$$

134
Theorem 24 (Reduction Preservation). A reduction in DC with induction
\[ t \cdot k \leadsto t' \cdot k' \]
induces a sequence of congruent reductions in DC with Mendler induction
\[ \bar{t} \cdot \bar{k} \leadsto* \bar{t'} \cdot \bar{k'} \]

Proof. Most cases are trivial given that the translation is a homomorphism.
\[
\langle t_1, t_2 \rangle \cdot \text{fst}[k] = \langle \bar{t}_1, \bar{t}_2 \rangle \cdot \text{fst}[\bar{k}] \leadsto \bar{t}_1 \cdot \bar{k}
\]
\[
\langle t_1, t_2 \rangle \cdot \text{snd}[k] = \langle \bar{t}_1, \bar{t}_2 \rangle \cdot \text{snd}[\bar{k}] \leadsto \bar{t}_2 \cdot \bar{k}
\]
\[
\bar{t}_1(t) \cdot [k_1, k_2] = t_1(\bar{t}) \cdot \left[ k_1, k_2 \right] \leadsto \bar{t}_1 \cdot \bar{k}_1
\]
\[
\bar{t}_2(t) \cdot [k_1, k_2] = t_2(\bar{t}) \cdot \left[ k_1, k_2 \right] \leadsto \bar{t}_2 \cdot \bar{k}_2
\]
\[
\text{not}(k) \cdot \text{not}(\bar{t}) = \text{not}(\bar{k}) \cdot \text{not}(\bar{t}) \leadsto \bar{t} \cdot \bar{k}
\]
The abstraction cases require, as usual, the substitutive property (lemma 20):
\[
\bar{a}(\bar{c}) = \bar{a} \cdot \bar{c}
\]
\[
\bar{a}(\bar{c}) \cdot \bar{k} = \bar{a} \cdot \bar{c} \cdot \bar{k}
\]
\[
\bar{a}(\bar{c}) \cdot \bar{k} \leadsto \bar{a} \cdot \bar{c} \cdot \bar{k}
\]
The only case left, induction, is the most interesting one.
\[
\text{in}^{F(X)}(t) \cdot \text{itr}_{A,a}^{F(X)}[k,l] = \min(\bar{t}) \cdot \text{mitr}_{\rho,a}^{F(X)} \left[ \text{kmmap}_{\beta}^{F(X)} \left[ x.((x \# \beta) \cdot \rho), \bar{k} \right], \bar{t} \right]
\]
\[
\leadsto \bar{t} \cdot \text{kmmap}_{\beta}^{F(X)} \left[ x.((x \# \beta) \cdot \rho), \bar{k} \right] \left[ \mu a. \left( \text{mitr}_{\rho,a}^{F(X)} \left[ \text{kmmap}_{\beta}^{F(X)} \left[ x.((x \# \beta) \cdot \rho), \bar{k}, \alpha \right], \alpha \right] \right) / \rho \right] \left[ \bar{t} / \alpha \right]
\]
\[
= \bar{t} \cdot \text{kmmap}_{\beta}^{F(X)} \left[ x.((x \# \beta) \cdot \rho), \bar{k} \right] \left[ x.((x \# \beta) \cdot \mu a. \left( \text{mitr}_{\rho,a}^{F(X)} \left[ \text{kmmap}_{\beta}^{F(X)} \left[ x.((x \# \beta) \cdot \rho), \bar{k}, \alpha \right], \alpha \right] \right) / \rho \right] \left[ \bar{t} / \alpha \right]
\]
\[
= \bar{t} \cdot \text{kmmap}_{\beta}^{F(X)} \left[ x.((x \# \beta) \cdot \rho), \bar{k} \right] \left[ \text{itr}_{A,a}^{F(X)}[k, \beta], \bar{t} / \alpha \right]
\]
\[
= \bar{t} \cdot \text{kmmap}_{\beta}^{F(X)} \left[ \text{itr}_{A,a}^{F(X)}[k, \beta], k \left[ \bar{t} / \alpha \right] \right]
\]
\[
= \bar{t} \cdot \text{kmmap}_{\beta}^{F(X)} \left[ \text{itr}_{A,a}^{F(X)}[k, \beta], k \left[ \bar{t} / \alpha \right] \right]
\]
\[
= t \cdot \text{kmmap}_{\beta}^{F(X)} \left[ \text{itr}_{A,a}^{F(X)}[k, \beta], k \left[ \bar{t} / \alpha \right] \right]
\]
\[
= t \cdot \text{kmmap}_{\beta}^{F(X)} \left[ \text{itr}_{A,a}^{F(X)}[k, \beta], k \left[ \bar{t} / \alpha \right] \right]
\]
5.6 Concluding Remarks

§5.24. Mendler Induction and Classical Logic  Thus concludes our introduction to Mendler Induction in a Classical setting. We have shown how to translate the ideas in chapter 4 into an LK style presentation, with left and right rules. The definition has some quite subtle details. Namely, it uses (implicit) substitution of co-variables to instantiate induction steps with the necessary data: the inductive call and the output continuation; and the need to use subtractive abstractions to encapsulate the inductive call in a way that can be reused by the induction step.

§5.25. The Good and the Bad  The resulting system is very expressive: it simulates the conversion rules of Mender induction; it can also be used to write a suitable mapping action and, thereby, much like its functional brethren, it can simulate (Classical) ordinary induction. There is a price to pay, however, in that the syntax very quickly becomes unwieldy. This is already a “feature” to some extent of both Mendler Induction and the Dual Calculus but which is much expanded when we put the two together. However, where we don’t lose out on—and this despite the expressiveness of the language—is in all the good reduction properties. This is the topic for the our next chapter.
Chapter 6

Reduction Properties of the Extension to Mendler-inductive Types

§6.1. Extended Proofs  Best viewed as a complement to chapter 3, this chapter shows that all the nice reductions properties satisfied by the second-order Dual Calculus are still extant in the extension to Mendler (co-) inductive types despite the increase in expressiveness of the calculus. In fact, the organization of the chapter is such that there should be a one-to-one mapping between proclamations there and here—so exact that the statements are often carbon copies of the original ones. For this reason, we shall continuously omit the cases for propositional and second-order types in their proofs and focus on their extension to co- and inductive cases. We will, then, refer the reader back to the original lemma, theorem, or proposition for the details.

§6.2. Structure in Full Use  The reader will have noticed that the much vaunted lattice structure played little rôle in chapter 3; at most, it helped separate certain concerns of the model, viz. using the joins and meets, we could show orthogonality for the subtractive action by arguing on a single arbitrary co-variable. This chapter is the *raison d'être* behind the overly complicated proofs introduced in its sibling. It is here that we put that lattice structure to good use, by arguing—by Tarski’s fix-point theorem (1955) and the inherent chain-closed property of lattices—that Mendler (co-) inductive types are greatest and least fix-points.

§6.3. Structure of the Chapter  So, again, we begin with the extension of the head reductions rules to their congruent closure; followed by the exact statement of the duality between induction and co-induction; the substitutive properties; and preservation of well-typing by the different forms of reduction. Then, we tackle the main technical result of this thesis: a realizability proof of strong normalization for the Dual Calculus with Mendler induction.

§6.4. The Final Hurdles  Despite all the ground work done so far, this will still be a non-trivial enterprise where several technicalities will come to the fore. Curiously, even though we have been studying a Classical language, our analysis has been Intuitionistic so far. Unfortunately, that will have to change; in general, we cannot prove that the completion \([\mathcal{L}]\) is continuous in the Scott sense (Scott, 1970; Plotkin, 1983) and so, we
REDUCTION PROPERTIES OF THE EXTENSION TO MENDLER-INDUCTIVE TYPES

\[ t_0 \rightarrow t_1 \Rightarrow \begin{cases} \min(t_0) \rightarrow \min(t_1) \\ mcoitr_{r,\alpha}(t_0, t) \rightarrow mcoitr_{r,\alpha}(t, t) \\ mcoitr_{\alpha}(t, t_0) \rightarrow mcoitr_{\alpha}(t, t_1) \end{cases} \]

\[ k_0 \rightarrow k_1 \Rightarrow \begin{cases} mout[k_0] \rightarrow mout[k_1] \\ mitr_{\rho,\alpha}[k, k_0] \rightarrow mitr_{\rho,\alpha}[k, k_1] \\ mitr_{\rho,\alpha}[k_0, k] \rightarrow mitr_{\rho,\alpha}[k_1, k] \end{cases} \]

Table 6.1: Parallel reduction for the Dual Calculus with Mendler induction

need to use a well-known result that Classically every monotone function on a chain complete partial order is continuous.

6.1 Parallel Reduction

§6.5. Extension of the Congruent Closure of Reduction The notion of reduction that we will consider here is the generalization of head reduction to its congruent closure—it is formally presented in table 6.1. Cuts inside terms and co-terms are allowed to reduce and here we need to explain how this is done inside the (co-) inductive operations. We recall that (as per the definition in table 3.1) we allow reductions inside lambda- and mu-abstractions (as opposed to certain reduction strategies for the lambda calculus).

6.2 Duality

§6.6. Self-duality The extension of the duality result (section 3.2) for the second-order Dual Calculus is given in table 6.2. Under this duality, co-induction serves as a model for induction, and vice-versa. As before, we prove three results. The first of these, a substitution lemma, shows the preservation of the model by substitution and is required by abstractions (which we do not deal with here) but also for the inductive operators themselves: they reduce by substitution, indeed double substitution. From this we then prove, in order, that head-reduction and parallel reduction preserve the duality.

Lemma 21 (Substitution). Substitution by terms and substitution by co-terms preserve the duality (\(\neg\)):

\[
\begin{align*}
(t[u/x])^\circ &= t^\circ[u^\circ/x^\circ] \\
(k[u/x])^\circ &= k^\circ[u^\circ/x^\circ] \\
(c[u/x])^\circ &= c^\circ[u^\circ/x^\circ] \\
(t[l/a])^\circ &= t^\circ[l^\circ/a^\circ] \\
(k[l/a])^\circ &= k^\circ[l^\circ/a^\circ] \\
(c[l/a])^\circ &= c^\circ[l^\circ/a^\circ]
\end{align*}
\]

1 Fun fact: a similar argument appears in the book by Troelstra and van Dalen (1988, top of p. 148) on Constructive Mathematics; unlike them, however, we are not able to escape our conundrum.
Theorem 25. The duality \((\neg \circ)\) preserves head reduction:

\[ c \leadsto d \implies c^\circ \leadsto d^\circ. \]

Proof. Extending the previous case analysis (Theorem 14) and by two uses on each case of the substitution lemma:

\[
\begin{align*}
(min(t) \cdot mitr_{\rho,a}[k,l])^\circ &= (mitr_{\rho,a}[k,l])^\circ \cdot (min(t))^\circ \\
&= mcoitr_{\rho,a}^{\circ}(k^\circ, l^\circ) \cdot mout[l^\circ] \\
&\leadsto (k^\circ \cdot (\lambda \alpha. (mcoitr_{\rho,a}^{\circ}(k^\circ, \alpha^\circ)) / \rho^o)^{l^\circ / \alpha^\circ}) \cdot t^\circ \\
&= (k^\circ \cdot (\lambda \alpha. ((mitr_{\rho,a}[k,a]^\circ)^{\rho^o})^{l^\circ / \alpha^\circ})) \cdot t^\circ \\
&= (k^\circ \cdot ((\mu \alpha. (mitr_{\rho,a}[k,a])^{\rho^o})^{l^\circ / \alpha^\circ})) \cdot t^\circ \\
&= (k \cdot (\mu \alpha. (mitr_{\rho,a}[k,a])^{\rho})^{l / \alpha}) \cdot t^\circ \\
&= (t \cdot k \cdot (\mu \alpha. (mitr_{\rho,a}[k,a])^{\rho})^{l / \alpha})^\circ
\end{align*}
\]

\[
\begin{align*}
(mcoitr_{r,x}(t,u) \cdot mout[k])^\circ &= (mout[k])^\circ \cdot (mcoitr_{r,x}(t,u))^\circ \\
&= min(k^\circ) \cdot mitr_{r,x}^{\circ}[t^\circ, u^\circ] \\
&\leadsto k^\circ \cdot (t^\circ [(\mu x^\circ. (mcoitr_{r,x}^{\circ}(t^\circ, x^\circ)) / r^\circ)]^{u^\circ / x^\circ}) \\
&= k^\circ \cdot (t^\circ [(\mu u^\circ. ((mcoitr_{r,x}(t,x))^{\rho}) / r^\circ)]^{u^\circ / x^\circ}) \\
&= k^\circ \cdot (t^\circ [(\lambda x^\circ. (mcoitr_{r,x}(t,x))^{\rho}) / r^\circ](u^\circ / x^\circ)) \\
&= k^\circ \cdot (t^\circ [(\lambda x^\circ. (mcoitr_{r,x}(t,x))^{\rho}) / r](u / x)^\circ \\
&= (t \cdot k \cdot (\lambda x^\circ. (mcoitr_{r,x}(t,x))^{\rho}) / r)[u / x] \cdot k^\circ
\end{align*}
\]
Theorem 26. The duality ($\sim^\circ$) preserves parallel reduction for terms $t$ and $u$, co-terms $k$ and $l$, and cuts $c$ and $d$:

$$
t \equiv u \implies t^\circ \equiv u^\circ \quad k \equiv l \implies k^\circ \equiv l^\circ
$$

$$
c \equiv d \implies c^\circ \equiv d^\circ
$$

Proof. By the extension of the proof by mutual induction on the definition of parallel reduction of theorem \ref{thm:head-reduction}. The base case now requires theorem \ref{thm:head-reduction} to account for the extra possible head reductions. \hfill \Box

### 6.3 Substitutivity

§6.7. Further Indistinguishability. The two notions of indistinguishability, (I) that substituting in a (co-) variable preserves reduction and (II) that substitution preserves reducts and redexes, still apply to the extension to Mendler inductive types. Substitutivity is central in the proofs of saturation, and they will take an even larger rôle in the current context: both properties shall be needed in the proofs of saturation of the orthogonal pairs used to interpret Mendler inductive types. Both head and parallel reduction satisfy the reduction properties as we now prove.

Theorem 27. For cuts $c$ and $d$, any terms $v$ and any co-terms $m$, and any co-variable $\alpha$ and variable $x$,

$$
c \equiv d \implies c[v/x] \equiv d[v/x] \quad c \equiv d \implies c[m/\alpha] \equiv d[m/\alpha]
$$

Proof. Straightforward extension of the proof of theorem \ref{thm:head-reduction}—care only required in the co-and iteration cases, where (up to $\alpha$ equivalence of phrases) the (co-) induction variables are assumed to be the distinct from the substitution (co-) variables and not bound on the (co-) terms being substituted in. \hfill \Box

Theorem 28. Substitution by terms $t$ and co-terms $k$ preserves parallel reduction on terms, co-terms, and cuts, respectively:

$$
t_0 \equiv t_1 \implies t_0[t/x] \equiv t_1[t/x] \quad t_0 \equiv t_1 \implies t_0[k/\alpha] \equiv t_1[k/\alpha]
$$

$$
k_0 \equiv k_1 \implies k_0[t/x] \equiv k_1[t/x] \quad k_0 \equiv k_1 \implies k_0[k/\alpha] \equiv k_1[k/\alpha]
$$

$$
c_0 \equiv c_1 \implies c_0[t/x] \equiv c_1[t/x] \quad c_0 \equiv c_1 \implies c_0[k/\alpha] \equiv c_1[k/\alpha]
$$

Proof. Again, the base case, substitutivity for head reduction, is essentially theorem \ref{thm:head-reduction}:

$$
c \equiv d \implies c[t/x] \equiv d[t/x] \implies c[t/x] \equiv d[t/x]
$$

For the induction step, the non-inductive cases are exactly a before. The extension of the proof to Mendler inductive types requires the same assumption on the bound variables as the proof of theorem \ref{thm:head-reduction}. \hfill \Box

140
SUBJECT REDUCTION

Lemma 22. The syntactic operators of the Dual Calculus with Mendler induction preserve the reflexive transitive closure of parallel reduction, namely:

\[ t_0 \Rightarrow^* t_1 \quad \Longleftrightarrow \quad \begin{cases} 
\min(t_0) \Rightarrow^* \min(t_1) \\
mcoitr_{r,x}(t_0, t) \Rightarrow^* mcoitr_{r,x}(t_1, t) \\
mcoitr_{t,x}(t_0) \Rightarrow^* mcoitr_{t,x}(t_1) \\
\end{cases} \]

\[ k_0 \Rightarrow^* k_1 \quad \Longleftrightarrow \quad \begin{cases} 
mitr_{\rho,\alpha}[k_0] \Rightarrow^* mitr_{\rho,\alpha}[k_1] \\
motr_{\rho,\alpha}[k_0] \Rightarrow^* motr_{\rho,\alpha}[k_1] \\
\end{cases} \]

Proof. Similar to lemma 2.

Theorem 29. Substituting in preserves parallel reduction.

\[ t_0 \Rightarrow^* t_1 \quad \Longleftrightarrow \quad \begin{cases} 
l[t_0/x] \Rightarrow^* l[t_1/x] \\
k[t_0/x] \Rightarrow^* k[t_1/x] \\
c[t_0/x] \Rightarrow^* c[t_1/x] \\
k_0 \Rightarrow^* k_1 \quad \Longleftrightarrow \quad \begin{cases} 
l[k_0/\alpha] \Rightarrow^* l[k_1/\alpha] \\
k[k_0/\alpha] \Rightarrow^* k[k_1/\alpha] \\
c[k_0/\alpha] \Rightarrow^* c[k_1/\alpha] \\
\end{cases} \]

Proof. By extending the induction of theorem 8. As by now usual, up to \( \alpha \)-equivalence of phrases, we assume that the substitution (co-) variables do not appear bound in the phrases where substitution is performed, nor are any of the free variables of the phrases being substituted in bound in the phrases under substitution.

6.4 Subject Reduction

§6.8. Hiding is Sound  The typing rules of Mendler induction combine both variable and type variable abstraction in one—the iterator—rule. But despite the additional complication, the proof of subject reduction—that well-typed phrases reduce to well-typed phrases—follows closely the proof of subject reduction for the second-order calculus: from a type variable abstraction lemma and a substitution lemma in the mold of lemmas 3 and 4 we prove subject reduction for head reduction; and from that the result follows for the congruent closure of the latter. This result is all the more telling in the context of Mendler induction because it shows the operational soundness of Mendler’s hiding trick—and motivates the use of Typing Systems to realize interesting operational invariants.

Lemma 23. For \( \vec{X} \) a vector \( X_1, \ldots, X_n \) of distinct type variables, let \( \Gamma(\vec{X}) \) and \( \Delta(\vec{X}) \) stand for context schemas where free occurrences of the variables \( \vec{X} \) in typing contexts \( \Gamma \) and \( \Delta \) are abstracted. Then, any valid typing of terms \( t \), co-terms \( k \) or cuts \( c \) on arbitrary \( \Gamma(\vec{X}) \) and \( \Delta(\vec{X}) \) yields a valid typing whenever the type variables \( \vec{X} \) are replaced by types \( \vec{T} = T_1, \ldots, T_n \) that do not have free any of the bound variables in the typescheme \( F(\vec{X}) \):

\[ \Gamma(\vec{X}) \vdash t : F(\vec{X}) \mid \Delta(\vec{X}) \quad \Rightarrow \quad \Gamma(\vec{T}) \vdash t : F(\vec{T}) \mid \Delta(\vec{T}) \]
REDUCTION PROPERTIES OF THE EXTENSION TO MENDLER-INDUCTIVE TYPES

\[
\Gamma(\vec{x}) \vdash k : F(\vec{x}) \vdash \Delta(\vec{x}) \implies \Gamma(\vec{t}) \vdash k : F(\vec{t}) \vdash \Delta(\vec{t})
\]

\[
\Gamma(\vec{x}) \vdash c \vdash \Delta(\vec{x}) \implies \Gamma(\vec{t}) \vdash c \vdash \Delta(\vec{t})
\]

Proof. Extending of the mutual induction proof of lemma 3, we see that the (co-) inductive cases, unsurprisingly, require a similar extension of the vectors of types and type variables, to account for the bound variable. Implicit is the assumption that these bound variables are also chosen not to conflict with the other bound type variables in the term. We omit the usual label for the induction step, which is the first step is the basis of each derivation. For once, we show the co-inductive case.

\[
\frac{(x : G(\vec{x}), r : G(\vec{x}) \rightarrow Y, \Gamma(\vec{x}) \vdash t : F(\vec{x}, Y) \vdash \Delta(\vec{x}))}{\Gamma(\vec{t}) \vdash mcoitr_{r,x}(t,u) : \nu Y.F(\vec{t}, Y) \vdash \Delta(\vec{t})}
\]

\[
\frac{(\Gamma(\vec{x}) \vdash k : F(\vec{x}, \nu Y.F(\vec{x}, Y)) \vdash \Delta(\vec{x}))}{\Gamma(\vec{t}) \vdash mout[k] : \nu Y.F(\vec{t}, Y) \vdash \Delta(\vec{t})}
\]

Corollary 5. Let \(X\) be a type variable that does not appear free in \(\Gamma\), nor in \(\Delta\). Then, any valid typing that depends on \(X\) yields a valid typing when \(X\) is replaced by any other type \(T\):

\[
\frac{\Gamma \vdash t : F(\Gamma) \vdash \Delta}{\Gamma \vdash t : F(T) \vdash \Delta}
\]

Proof. Follows from lemma 3 by noting that, since \(X\) does not appear free in \(\Gamma\) (resp. \(\Delta\)), \(\Gamma(T) = \Gamma\) (resp. \(\Delta(T) = \Delta\))

Remark 4. Like those in remark 1, typings of (co-) inductive syntactic operators also determine the structure of their admissible types:

\[
\frac{\Gamma \vdash \text{min}(t) : T \vdash \Delta}{\implies T = \mu X.F(X) \text{ and } \Gamma \vdash t : F(\mu X.F(X)) \vdash \Delta}
\]

\[
\frac{\Gamma \vdash \text{mitr}_{r,\alpha}[k,l] : T \vdash \Delta}{\implies T = \mu X.F(X), \Gamma \vdash k : F(X) \vdash \Delta, \rho : X \rightarrow A, \alpha : A \text{ and } \Gamma \vdash l : A \vdash \Delta}
\]

\[
\frac{\Gamma \vdash \text{mcoitr}_{r,x}(t,u) : T \vdash \Delta}{\implies T = \nu X.F(X), \ x : A, r : A \rightarrow X, \Gamma \vdash t : F(X) \vdash \Delta \text{ and } \Gamma \vdash u : A \vdash \Delta}
\]

\footnote{This is exactly the argument of corollary 3}
SUBJECT REDUCTION

\[ \Gamma \vdash \text{mout}[k] : T \vdash \Delta \]
\[ \implies T = vX.F(X) \text{ and } \Gamma \vdash t : F(vX.F(X)) \vdash \Delta \]

for some \( A, B \) and \( F(X) \) with \( X \) not free in the relevant typing (co-) contexts.

Lemma 24. Substitutions on well-typed phrases induce well-typed phrases:

\[ \Gamma \vdash t : A \vdash \Delta \implies \forall \Gamma', \Delta', \sigma : (\Gamma \vdash \Delta) \mapsto (\Gamma' \vdash \Delta') . \Gamma' \vdash t[\sigma] : A \vdash \Delta' \]
\[ \Gamma \vdash k : A \vdash \Delta \implies \forall \Gamma', \Delta', \sigma : (\Gamma \vdash \Delta) \mapsto (\Gamma' \vdash \Delta') . \Gamma' \vdash k[\sigma] : A \vdash \Delta' \]
\[ \Gamma \vdash c \vdash \Delta \implies \forall \Gamma', \Delta', \sigma : (\Gamma \vdash \Delta) \mapsto (\Gamma' \vdash \Delta') . \Gamma' \vdash c[\sigma] \vdash \Delta' \]

where, up to \( \alpha \)-conversion of types, we assume no bound type variables in \( A \) appear free in \( \Gamma' \) or \( \Delta' \).

Proof. By extension of the induction in lemma \ref{lemma23}. The co- and iteration rules require picking a type variable that is not free in the co-domain of \( \Gamma' \) and \( \Delta' \).

From the lemmas above we can show that head reduction preserves typings:

Theorem 30 (Head Subject Reduction). For any two cuts \( c \rightsquigarrow d \),

\[ \Gamma \vdash c \vdash \Delta \implies \Gamma \vdash d \vdash \Delta. \]

Proof. Two more cases to consider relative to the proof of theorem \ref{theorem28}. For induction, take a well-typed redex

\[ \Gamma \vdash \text{min}(t) \bullet \text{mitr}_{\rho,\alpha}[k,l] \vdash \Delta. \]

As per remark \ref{remark4} we have that

\[ \Gamma \vdash t : F(\mu X.F(X)) \vdash \Delta, \]
\[ \Gamma \vdash k : F(X) \vdash \Delta, \rho : X \rightarrow A, \alpha : A, \]
\[ \Gamma \vdash l : A \vdash \Delta; \]

whence, by lemma \ref{lemma23} on the instantiation of free type variables in typing judgements, also

\[ \Gamma \vdash k : F(\mu X.F(X)) \vdash \Delta, \rho : \mu X.F(X) \rightarrow A, \alpha : A. \]

Choose a fresh \( \beta \); then

\[ \Gamma \vdash k : F(X) \vdash \Delta, \rho : X \rightarrow A, \alpha : A \]
\[ \Gamma \vdash k : F(X) \vdash \Delta, \beta : A, \rho : X \rightarrow A, \alpha : A \]
\[ \Gamma \vdash \beta : A \vdash \Delta, \beta : A \]

\[ \Gamma \vdash \text{mitr}_{\rho,\alpha}[k,\beta] : \mu X.F(X) \vdash \Delta, \beta : A \]
\[ \Gamma \vdash \mu \beta.(\text{mitr}_{\rho,\alpha}[k,\beta]) : \mu X.F(X) \rightarrow A \vdash \Delta \]

By the substitution lemma, we conclude

\[ \Gamma \vdash k : F(\mu X.F(X)) \vdash \Delta, \rho : \mu X.F(X) \rightarrow A, \alpha : A \]
\[ \implies \Gamma \vdash k[\mu \beta.(\text{mitr}_{\rho,\alpha}[k,\beta]) / \rho] : F(\mu X.F(X)) \vdash \Delta, \alpha : A \]
REDUCTION PROPERTIES OF THE EXTENSION TO MENDLER-INDUCTIVE TYPES

$$\Gamma \vdash k[\mu \beta. (\text{mitr}_{\rho, \alpha}(k, \beta))/\rho][l/\alpha] : F(\mu X. F(X)) \vdash \Delta$$

$$\Gamma \vdash t \cdot k[\mu \beta. (\text{mitr}_{\rho, \alpha}(k, \beta))/\rho][l/\alpha] \vdash \Delta$$

which (up to $\alpha$-equivalence of the co-term) is the intended typing. We omit here the co-inductive case as it is similar. □

**Theorem 31 (Subject Reduction)**. Parallel reduction in DC preserves typing:

$$\Gamma \vdash t : A | \Delta \text{ and } t \leadsto u \implies \Gamma \vdash u : A | \Delta$$

$$\Gamma \vdash k : A \vdash \Delta \text{ and } k \leadsto l \implies \Gamma \vdash l : A \vdash \Delta$$

$$\Gamma \vdash c \vdash \Delta \text{ and } c \leadsto d \implies \Gamma \vdash c \vdash \Delta$$

Proof. By extending the induction on the definition of parallel reduction of theorem 10 with inductive and co-inductive operations. The base case for the extension to Mendler inductive types is now given by preservation for head reduction (theorem 30). The remaining cases are borne by the coupling of remark 4 and the induction hypothesis. □

### 6.5 Strong Normalization

§6.9. Outline  The main result of this thesis is the establishment that the extension of the second-order Dual Calculus with Mendler (co-) inductive types does not have any non-terminating cuts. This entails: extending the sets of syntax, lattices $\mathcal{O}P$ and $\mathcal{O}N\mathcal{P}$ and associated actions to account for the new syntactic constructs; capturing the reduction behaviour of Mendler (co-) iterations at the set level; showing how to account for the arbitrary variance of the type-schemes associated with the (co-) iterators and thereby providing fix-point interpretations of (co-) inductive types; and showing that the model is adequate with respect to the reduction rules of Mendler induction. There are, therefore, quite a few steps to cover, complicated further by two issues: the peculiar nature of the reduction rule of Mendler (co-) induction; and the fact that the mappings that we can derive from the interpretation of arbitrary typeschemes are not in general monotone, let alone continuous. The first complication will force us to forsake the strict use of algebraic principles in our proofs; the second of continuing with our Intuitionistic study of a Classical language. Despite not being able to construct the interpretation for inductive types as a limit of an ascending chain—as typically done in domain theory (Scott, [1970]; Plotkin, [1983])—we will still be able to extract an induction principle for our inductive types. This will require us to move from increasing chains to increasing transfinite chains; and, thence, Classically finding the necessary fix-points.

So, without further ado...

### 6.5.1 Sets and Saturation

§6.10. Sets of Syntax  First and foremost, as the definition of the language changes, the sets of phrases in play need to change accordingly to accommodate the additional
STRONG NORMALIZATION

syntactic elements. In particular the sets of introduction terms and eliminator co-terms
§3.16 need to be expanded with

\[ \mathcal{IT} = \ldots , \text{min}(t), \text{mcoitr}_{r,x}(t,u) \quad \text{and} \quad \mathcal{EK} = \ldots , \text{mitr}_{\rho,a}[k,l], \text{mout}[k]. \]

The sets \( \mathcal{SN}\mathcal{T} \), \( \mathcal{SN}\mathcal{K} \), and \( \mathcal{SN} \) maintain their definitions relative to the expanded syntax
of inductive and co-inductive operators.

§6.11. Syntactic Actions  The syntactic actions associated with inductive operators follow essentially the same pattern as those of the second order calculus—e.g, given a set of
terms we simply apply the constructor to each of the terms.

\[
\begin{align*}
\text{min}\langle T \rangle & \equiv \{ \text{min}(t) \mid t \in T \} \\
\text{mitr}_{\rho,a}[K,L] & \equiv \{ \text{mitr}_{\rho,a}[k,l] \mid k \in K, l \in L \} \\
\text{mcoitr}_{r,x}(T,U) & \equiv \{ \text{mcoitr}_{r,x}(t,u) \mid t \in T, u \in U \} \\
\text{mout}[K] & \equiv \{ \text{mout}[k] \mid k \in K \}
\end{align*}
\]

Again, these are but the image of the respective (injective) operators and therefore are
monotonic on sets under inclusion, and preserve unions and non-empty intersections.

§6.12. Congruent Reduction and Saturation  As a result of the definitions of the actions
as images of the operations, we can easily describe parallel reductions for sets made out
of these operators:

\[
\begin{align*}
\mathllbracket \text{min}(T) \rrbracket & = \text{min}\langle \mathllbracket (T) \rangle \rangle \\
\mathllbracket \text{mitr}_{\rho,a}[K,L] \rrbracket & = \text{mitr}_{\rho,a}\langle \mathllbracket (K) \rangle, \mathllbracket (L) \rangle \rangle \\
\mathllbracket \text{mcoitr}_{r,x}(T,U) \rrbracket & = \text{mcoitr}_{r,x}\langle \mathllbracket (T) \rangle, \mathllbracket (U) \rangle \rangle \\
\mathllbracket \text{mout}[K] \rrbracket & = \text{mout}\langle \mathllbracket (K) \rangle \rangle \\
\end{align*}
\]

Their saturation then follows easily.

Lemma 25. Let \( T \) and \( U \) be saturated sets of terms, \( K \) and \( L \) be saturated sets of co-terms.
Inductive and co-inductive terms and co-terms built out of them and the inductive and
co-inductive introductions and eliminations are saturated:

\[
\begin{align*}
\mathllbracket \text{min}(T) \rrbracket & \subseteq \text{min}(T) \\
\mathllbracket \text{mitr}_{\rho,a}[K,L] \rrbracket & \subseteq \text{mitr}_{\rho,a}[K,L] \\
\mathllbracket \text{mcoitr}_{r,x}(T,U) \rrbracket & \subseteq \text{mcoitr}_{r,x}(T,U) \\
\mathllbracket \text{mout}[K] \rrbracket & \subseteq \text{mout}[K]
\end{align*}
\]

Proof. Akin to that of lemma \( \mathbf{\S} \).
§6.13. Mendler Restriction  Mendler (co-) induction reduces by substitution. As for abstractions, then, we need a restriction operation that acts as a guard for the phrases suitable to be put in by substitution. However, the reduction of any particular instance of the Mendler iterator is not as homogeneous as the reduction of an abstraction: the latter must accept for substitution all suitable phrases of the right type; but the induction co-variable in an induction step $k$ is only ever substituted by $\mu a.(mitr_{\rho,a}[k,a])$. The two restrictions for Mendler (co-) induction are defined as

\[
K/_{\rho}L \triangleq \{ k \in SN\mathcal{K} \mid \text{for all } l \in L, k[\mu a.(mitr_{\rho,a}[k,a])/\rho][l/a] \in K \}
\]

\[
T/_{x}U \triangleq \{ t \in SN\mathcal{T} \mid \text{for all } u \in U, t[\lambda x. (mcoitr_{r,x}(t,x))/r][u/x] \in T \}.
\]

The above technicality has major implications in our effort to describe the model as an algebra on sets—we shall not enjoy the usual adjoint situation with substitution. One advantage of this would be that we could simply reuse the proof of theorem 11 to determine the saturation of restriction. Instead, we must prove it directly and even strengthen the proof: to guarantee saturation we must use the second substitutivity property (theorem 29) and we also must require that congruent reductions be performed inside subtractive and lambda abstractions.

**Lemma 26** (Saturation for Mendler Restrictions). Let $T$ be a set of terms, and $K$ a set of co-terms—both of them saturated—and let $U$ and $L$ be any set of term and any set of co-terms, respectively. For any (distinct) variables $r$ and $x$, and co-variables $\rho$ and $\alpha$,

\[
K/_{\rho}L \quad \text{and} \quad T/_{x}U
\]

are saturated.

**Proof.** We focus here only on the restriction for induction. Take $k \rightsquigarrow k'$; this implies $mitr_{\rho,a}[k,a] \rightsquigarrow mitr_{\rho,a}[k',a]$ and thence, by congruent reduction inside catches,

\[
\mu a.(mitr_{\rho,a}[k,a]) \rightsquigarrow \mu a.(mitr_{\rho,a}[k',a]).
\]

Now, if, additionally, $k$ is in the restriction,

\[
k \in K/_{\rho}L \implies \text{for any } l \in L k[\mu a.(mitr_{\rho,a}[k,a])/\rho][l/a] \in K
\]

\[
\implies \text{for any } l \in L k'[\mu a.(mitr_{\rho,a}[k,a])/\rho][l/a] \in K
\]

\[
\implies \text{for any } l \in L k'[\mu a.(mitr_{\rho,a}[k',a])/\rho][l/a] \in K
\]

\[
k' \in K/_{\rho}L
\]

**Lemma 27** (Preservation of (Head) Orthogonality). Take $T, U \subseteq SN\mathcal{T}$ to be sets of terms, $K, L \subseteq SN\mathcal{K}$ to be sets of co-terms, and assume that $T \cdot K \subseteq SN$; it then follows that

\[
[\rightsquigarrow](\text{min}(T) \cdot mitr_{\rho,a}[K/_{\rho}L,L]) \subseteq SN \quad (\rho \neq a \in \text{Covar})
\]

\[
[\rightsquigarrow](mcoitr_{r,x}(T/_{x}U,U) \cdot mout[K]) \subseteq SN \quad (r \neq x \in \text{Var}).
\]

146
STRONG NORMALIZATION

Proof.

\[
\begin{align*}
\min(t) \in \min(T) \quad & \text{and} \quad \mitr_{\rho,\alpha}[k, l] \in \mitr_{\rho,\alpha}[K/\rho L, L] \\
\implies t \in T, k \in K/\rho L, l \in L \\
\implies t \in T, \text{for any } l' \in L. k [\mu a.(\mitr_{\rho,\alpha}[k, a]) / \rho] [l'/a] \in K, l \in L \\
\implies t \in T, k [\mu a.(\mitr_{\rho,\alpha}[k, a]) / \rho] [l/a] \in K \\
\implies t \bullet k [\mu a.(\mitr_{\rho,\alpha}[k, a]) / \rho] [l/a] \in SN' \\
\implies \sigma \min(t) \bullet \mitr_{\rho,\alpha}[k, l] \subseteq SN'
\end{align*}
\]

\[
\begin{align*}
mcoitr_{r,x}(t, u) \in mcoitr_{r,x}(T/\rho U, U) \quad & \text{and} \quad \mout[k] \in \mout[K] \\
\implies t \in T/\rho U, u \in U, k \in K \\
\implies \text{for any } u' \in U. t [\lambda x.(mcoitr_{r,x}(t, x)) / r] [u'/x] \in T, u \in U, k \in K \\
\implies t [\lambda x.(mcoitr_{r,x}(t, x)) / r] [u/x] \in T, k \in K \\
\implies t [\lambda x.(mcoitr_{r,x}(t, x)) / r] [u/x] \bullet k \in SN' \\
\implies \sigma mcoitr_{r,x}(t, u) \bullet \mout[k] \subseteq SN'
\end{align*}
\]

\[\square\]

Lemma 28 (Preservation of Orthogonality). Take \(T, U \subseteq SN'\) to be saturated sets of strongly normalizing terms, \(K, L \subseteq SN'\) to be saturated sets of strongly normalizing co-terms, and assume that \(T \bullet K \subseteq SN\); it then follows that

\[
\begin{align*}
\min(T) \bullet \mitr_{\rho,\alpha}[K/\rho L, L] & \subseteq SN & (\rho \neq \alpha \in Covar) \\
mcoitr_{r,x}(T/\rho U, U) \bullet \mout[K] & \subseteq SN & (r \neq x \in Var).
\end{align*}
\]

Proof. Lemma 6—stating that saturation and preservation of orthogonality by head reduction are sufficient to establish preservation of orthogonality by parallel reduction—still applies, whence lemma 27 suffices to establish the result. \[\square\]

6.5.2 Orthogonal Pairs and Their Actions

§6.14. Lattices of Syntax  Going one level up in terms of structure, we find ourselves at the level of lattices; specifically those of orthogonal and orthogonal normal pairs. Propositions 8 and 9 where we laid out explicitly the lattice structure of the aforementioned sets, depended only on the definitions of \(SN'\), \(SN'\), and \(SN'\). As such, the proofs and the propositions still apply for the extension—we will, therefore, omit repeating those statements and jump straight to the actions between \(OP\) and \(ONP\).
§6.15. Mendler Pairing  Fixing the orthogonal sets $T$ and $K$ in the previous constructions, one obtains orthogonal normal pairs for particular combinations of their parameter (co-) variables and continuation/seed sets; however, it is not hard to generalize this to every possibility by taking a union over all possible parameters. So, take $P \in \mathcal{OP}$ to be our base orthogonal pair (think interpretation of a typescheme); we define the pairing for Mendler inductive and co-inductive types to be, respectively,

$$\text{MuP} : \mathcal{OP} \rightarrow \mathcal{ONP}$$

$$\text{MuP}(P) = \left\{ \min\langle (P)^\top \rangle, \bigcup_{Q \in \mathcal{OP}} \text{mitr}_{\rho, \alpha} \left[ (P)^K_{/\alpha} / (Q)^K_{/\alpha}, (Q)^K_{/\alpha} \right] \right\}$$

$$= \bigcap_{Q \in \mathcal{OP}} \min\langle (P)^\top \rangle, \bigcup_{Q \in \mathcal{OP}} \text{mitr}_{\rho, \alpha} \left[ (P)^K_{/\alpha} / (Q)^K_{/\alpha}, (Q)^K_{/\alpha} \right]$$

$$= \bigwedge_{Q \in \mathcal{OP}} \left( \min\langle (P)^\top \rangle, \text{mitr}_{\rho, \alpha} \left[ (P)^K_{/\alpha} / (Q)^K_{/\alpha}, (Q)^K_{/\alpha} \right] \right) \in \mathcal{ONP}$$

$$\text{NuP} : \mathcal{OP} \rightarrow \mathcal{ONP}$$

$$\text{NuP}(P) = \left\{ \bigcup_{Q \in \mathcal{OP}} \text{mcoitr}_{r, x} \left[ (P)^\top_{/x} / (Q)^\top_{/x}, (Q)^\top_{/x} \right], \text{mout}[ (P)^\top_{/x} ] \right\}$$

$$= \bigcup_{Q \in \mathcal{OP}} \text{mcoitr}_{r, x} \left[ (P)^\top_{/x} / (Q)^\top_{/x}, (Q)^\top_{/x} \right], \bigcap_{Q \in \mathcal{OP}} \text{mout}[ (P)^\top_{/x} ]$$

$$= \bigvee_{Q \in \mathcal{OP}} \left( \text{mcoitr}_{r, x} \left[ (P)^\top_{/x} / (Q)^\top_{/x}, (Q)^\top_{/x} \right], \text{mout}[ (P)^\top_{/x} ] \right) \in \mathcal{ONP}$$

That they form orthogonal normal pairs follows from the fact that each element in the meet is orthogonal for orthogonal $P \in \mathcal{OP}$ (by lemma 28) and the definition of orthogonal pairs), and made at the outermost level of constructors/eliminators.

§6.16. Monotonization  Given the syntactic actions, one might be tempted to define the action for inductive types as the least fix-point of some function $f$, representing the action of the functor, followed by the constructor/eliminator:

$$\text{lfp}(\text{MuP} \circ f).$$

This, however, is not sufficient for our purposes. The typeschemes that parametrize our version of Mendler induction need not—in any way—represent functors and be
co-variant. Correspondingly, we need to account for the fact that the functions that parametrize the inductive action may not be monotone. Without monotonicity we cannot establish, in general, the existence of fix-points—we will have to find the fix-points elsewhere. The straightforward—albeit not obvious—solution (Uustalu and Vene, 1999; Matthes, 1999a) is to use monotone functions\(^3\) that more closely approximate our \(f\)—there are two such that are canonical.

To illustrate the method consider the function \(f(x) = x^3 - 3x\); if we take the usual ordering on the reals, it is clearly not monotone due to the dip in the range \(-1 < x < 1\).

To obtain—canonically—monotonicity, we need to ensure one of two things: that the image of the function at any point \(a\) is bigger than the image at any point \(b\) that precedes it; or that the image of the function at any point \(a\) is smaller than the image of the function for every point \(b\) that follows \(a\). Applying both forms of reasoning to the graph of the aforementioned function yields

To the first one, sitting above the initial function, we term its monotone extension; to the second, sitting below, we term its monotone restriction.

For functions whose image lies on a lattice\(^4\)—in particular the complete lattice \(\Omega N P\)—

---

\(^3\)The cognoscenti will recognize here the left and right Kan extensions of a function over the identity inclusion (see e.g. Mac Lane, 1998, chap. X)

\(^4\)Note that reflexivity prevents us from having to consider empty lubs and glbs
we can always apply the above method to yield monotone functions. The extension is given by

\[ [f] x \overset{\text{def}}{=} \bigvee_{y \leq x} f y = \left( \bigcup_{y \leq x} (f y)^T, \bigcap_{y \leq x} (f y)^K \right), \]

whereas the restriction of \( f \) is given by the dual

\[ [f] x \overset{\text{def}}{=} \bigwedge_{x \leq y} f y = \left( \bigcap_{x \leq y} (f y)^T, \bigcup_{x \leq y} (f y)^K \right). \]

Lemma 29 (Characterization of Monotonizations). If \( f \) is an arbitrary endofunction on a complete lattice \( L \) then its monotone extension, \( [f] : L \to L \), is the least monotone function above \( f \) while its monotone restriction \( [f] : L \to L \) is the greatest monotone function below \( f \).

Proof. Easily, for any \( x \in L \) and by the universal properties of the lattice constructions

\[ f x \leq \bigvee_{y \leq x} f y = [f] x \quad \text{and} \quad f x \geq \bigwedge_{y \leq x} f y = [f] x. \]

For monotonicity, we need to prove that for \( x \leq y \), the operations retain the order:

\[ [f] x \leq [f] y \quad \text{and} \quad [f] x \leq [f] y. \]

Let us consider the extension; for any \( z \leq x \), we know that \( z \leq y \) by transitivity; whence

\[ f z \leq \bigvee_{z \leq y} f z, \]

which is to say that term on the right is an upper bound for \( z \leq x \) and, therefore

\[ \bigvee_{z \leq x} f z \leq \bigvee_{z \leq y} f z \]

which is, by definition, equivalent to saying

\[ [f] x \leq [f] y. \]

The result for the monotone restrictions follows dually.

§6.17. Fix-points One particular instance this lemma applies to is endofunctions on the lattice \( \mathcal{ONP} \). Pairing it with Tarski’s fix-point theorem \([1955]\) yields a result of the utmost consequence.

Corollary 6. The monotone extension \( [f] : \mathcal{ONP} \to \mathcal{ONP} \) and the monotone restriction \( [f] : \mathcal{ONP} \to \mathcal{ONP} \) of any endofunction \( f \) on orthogonal normal pairs both have fixpoints. Further, the sets of their respective fix-points form themselves sub-lattices of \( \mathcal{ONP} \), with greatest and least elements.
§6.18. Inductive Actions. For induction we will use the one that contains all the possible terms—namely, those of the form \( \text{min}(t) \); for co-induction, we privilege the inclusion of all (sensible) continuations—those of the form \( \text{mout}(k) \). The inductive action takes the least fix-point of the monotone extension in \( \mathcal{ONP} \) of the “inductive step”—this being comprised of the interpretation of a typescheme (as a function \( f \)) followed by the completion with the inductive operators. The action for co-induction is dual: we compose \( f \) and the completion with the co-inductive constructor and eliminator, and then take the greatest fix-point of the monotonic restriction:

\[
\mu f \equiv \text{lfp}([\text{MuP} \circ f]) \quad \text{and} \quad \nu f \equiv \text{gfp}([\text{NuP} \circ f]).
\]

Theorem 32 (Orthogonality for Actions). For any arbitrary function \( f : \mathcal{ONP} \to \mathcal{OP} \),

\[
\mu f \in \mathcal{ONP} \quad \text{and} \quad \nu f \in \mathcal{ONP}.
\]

Proof. For \( f : \mathcal{ONP} \to \mathcal{OP} \), both \( \text{MuP} \circ f \) and \( \text{NuP} \circ f \) are functions \( \mathcal{ONP} \to \mathcal{ONP} \); then corollary \( 6 \) witnesses the existence of the claimed fix-points for

\[
[\text{MuP} \circ f] \quad \text{and} \quad [\text{NuP} \circ f].
\]

\[ \square \]

6.5.3 Orthogonal Interpretations

§6.19. Interpretations. Given a type \( T \) and a mapping \( \gamma : \text{ftv}(T) \to \mathcal{ONP} \) (the context) define the following normal interpretations by mutual induction on the structure of \( T \)

\[
[\mu X.F(X)](\gamma) = \mu(\langle F(X)\rangle(\gamma[X \mapsto -]))
\]

\[
[\nu X.F(X)](\gamma) = \nu(\langle F(X)\rangle(\gamma[X \mapsto -]))
\]

As before, the orthogonal interpretation is given by

\[
\langle T \rangle(\gamma) = \llbracket T \rrbracket(\gamma)
\]

Theorem 33 (Well-definedness). For any type \( T \) of the Mendler inductive extension of DC, and for any suitable interpretation context \( \gamma (\text{ftv}(T) \subseteq \text{dom}(\gamma)) \),

\[
[\llbracket T \rrbracket(\gamma) \in \mathcal{ONP} \quad \text{and} \quad \langle T \rangle(\gamma) \in \mathcal{OP}
\]

Proof. We extend the proof of the corresponding theorem for the second-order dual calculus (theorem \( 13 \)) for inductive types \( \mu X.F(X) \) and co-inductive types \( \nu X.F(X) \). By theorem \( 32 \) it suffices to show that the extended interpretation of the typescheme \( F(X) \) is a function \( \mathcal{ONP} \to \mathcal{ONP} \); for, then,

\[
[\mu X.F(X)](\gamma) = \mu(\langle F(X)\rangle(\gamma[X \mapsto -])) \in \mathcal{ONP}
\]

and

\[
[\nu X.F(X)](\gamma) = \nu(\langle F(X)\rangle(\gamma[X \mapsto -])) \in \mathcal{ONP}.
\]
This follows immediately from the induction hypothesis: for any \( N \in \mathcal{O}\mathcal{N}\mathcal{P} \),

\[
\downarrow F(X)\uparrow (\gamma[X \to N]) \in \mathcal{O}\mathcal{P}
\]
as the free type variables in \( F(X) \) are those in \( \mu X.F(X) \) plus \( X \), as needed—and similarly

\[
\nu X.F(X).
\]

Lemma 30. The two interpretations are still related (lemma [10]) by

\[
\begin{align*}
\operatorname{Var}, ([T]([\gamma]))^T & \subseteq ([\downarrow T\uparrow([\gamma]))^T \\
\operatorname{Covar}, ([T]([\gamma]))^K & \subseteq ([\downarrow T\uparrow([\gamma]))^K
\end{align*}
\]

Proof. The definition of the orthogonal interpretation is exactly the same as before, and so we can reuse the original proof.

Lemma 31 (Weakening). Let \( N \in \mathcal{O}\mathcal{N}\mathcal{P} \) and \( Y \notin \text{ftv}(T) \):

\[
\begin{align*}
[T](\gamma[Y \mapsto N]) & = [T](\gamma) \\
\downarrow T\uparrow(\gamma[Y \mapsto N]) & = \downarrow T\uparrow(\gamma)
\end{align*}
\]

Proof. We provide here only the calculations for the co- and inductive cases, which were not contemplated in lemma [11]. The same observation on bound type variables which we found before in the case of quantified types—that they can be chosen so as to not coincide with \( Y \)—applies, and thence

\[
\begin{align*}
\mu X.F(X)(\gamma[Y \mapsto N]) & = \mu(\downarrow F(X)\uparrow(\gamma[Y \mapsto N][X \mapsto -])) \\
& = \mu(\downarrow F(X)\uparrow(\gamma[X \mapsto -][Y \mapsto N])) \\
& = \mu(\downarrow F(X)\uparrow(\gamma[X \mapsto -])) & \text{(IH)} \\
& = [\mu X.F(X)](\gamma)
\end{align*}
\]

\[
\begin{align*}
\nu X.F(X)(\gamma[Y \mapsto N]) & = \nu(\downarrow F(X)\uparrow(\gamma[Y \mapsto N][X \mapsto -])) \\
& = \nu(\downarrow F(X)\uparrow(\gamma[X \mapsto -][Y \mapsto N])) \\
& = \nu(\downarrow F(X)\uparrow(\gamma[X \mapsto -])) \\
& = [\nu X.F(X)](\gamma)
\end{align*}
\]

Lemma 32 (Substitution). Let \( T, T' \) be types and \( \text{ftv}(T, T') - \{Y\} \subseteq \text{dom}(\gamma) \):

\[
\begin{align*}
[T[T'/Y]](\gamma) & = [T](\gamma[Y \mapsto [T'](\gamma)]) \\
\downarrow T[T'/Y]\uparrow(\gamma) & = \downarrow T\uparrow(\gamma[Y \mapsto [T'](\gamma)])
\end{align*}
\]
STRONG NORMALIZATION

Proof. To extend the previous proof by mutual induction for inductive types \( \mu X. F(X) \) and \( \nu X. F(X) \) — \( X \) is bound so it can be chosen both not free in \( T' \) and different from \( Y \) — we calculate

\[
\begin{align*}
\left[ \mu X. F(X) \right][T'/Y] &= \left[ \mu X. F(X)[T'/Y] \right](\gamma) \\
&= \mu (\downarrow F(X)[T'/Y] \uparrow(\gamma[X \mapsto -])) \\
&= \mu (\downarrow F(X) \uparrow(\gamma[X \mapsto -][Y \mapsto [T'](\gamma[X \mapsto -])))) \\
&= \mu (\downarrow F(X) \uparrow(\gamma[X \mapsto -][Y \mapsto [T'](\gamma)])) \\
&= \mu (\downarrow F(X) \uparrow(\gamma[Y \mapsto [T'](\gamma)])) \\
&= \left[ \mu X. F(X) \right](\gamma[Y \mapsto [T'](\gamma)])
\end{align*}
\]

\[
\begin{align*}
\left[ \nu X. F(X) \right][T'/Y] &= \left[ \nu X. F(X)[T'/Y] \right](\gamma) \\
&= \nu (\downarrow F(X)[T'/Y] \uparrow(\gamma[X \mapsto -])) \\
&= \nu (\downarrow F(X) \uparrow(\gamma[X \mapsto -][Y \mapsto [T'](\gamma)])) \\
&= \nu (\downarrow F(X) \uparrow(\gamma[Y \mapsto [T'](\gamma)])) \\
&= \left[ \nu X. F(X) \right](\gamma[Y \mapsto [T'](\gamma)])
\end{align*}
\]

6.5.4 Conservation for (Co-) Inductive Types

§6.20. Lack of Continuity The proof of conservation for inductive types is considerably more complicated than for propositional and quantified types — so much so that the proof we will give here will, in fact, not be constructive. For inductive types, on the term side, all we need is to appeal to the fix-point property; the key problem lies with the co-term side of the interpretation in that (roughly) terms of the form \( \mu \rho; \alpha; k; l \) depend on terms of the form \( \mu \alpha; (\mu \rho; \alpha; k; l) \), whose interpretation in turn will depend on \( \mu \rho; \alpha; k; l \), for all \( l \) in the co-term side of the relevant orthogonal pair — a seemingly inescapable circularity. The usual way of solving this problem is to prove that the interpretation of the typescheme under consideration is — not only monotonic but also — least upper bound preserving and preserves, in this case, the property of being inhabited by \( \mu \rho; \alpha; k; l \). Unfortunately, the monotonization constructions do not lift to the level of continuity. An alternative would be to use the least pre-fix point property as afforded to us by Tarski’s fix-point theorem; unfortunately, this least pre-fix point property does not seem to be strong enough to get us our result.

§6.21. A Classical Characterization The least fix-point of Tarski’s fix-point theorem does have, Classically, a description as a least upper bound. To derive the fix-point principle we need to take the least upper bound of a transfinite chain. This theorem was
REDUCTION PROPERTIES OF THE EXTENSION TO MENDLER-INDUCTIVE TYPES

stated (for complete partial orders) by Davey and Priestley [2002, theorem 4.15, proof outline in exercise 4.13]. It is paramount to us, and so we shall now prove the result in detail. We do not need directly that the fix-point be least (§6.22) and so we will state the theorem only for some fix-point.

Proposition 16 (Transfinite Chains of Monotone Functions). For any monotone endo-function \( f : \mathcal{ONP} \rightarrow \mathcal{ONP} \), the mapping \( d_\alpha \), parametrized by ordinals \( \alpha \in \text{Ord} \), given by

\[
\begin{align*}
d_{\beta+1} &= f \ d_\beta \\
d_\lambda &= \bigvee_{\beta < \lambda} d_\beta
\end{align*}
\]

for successor ordinals

for limit ordinals

determines an increasing chain:

\( \alpha \leq \beta \implies d_\alpha \leq d_\beta. \)

In particular, for the limit ordinal zero the definition devolves into

\( d_0 = \bigvee_{\beta \in \emptyset} f \beta = \bot. \)

Proof. To prove that \( d \) is a chain, as claimed, we use transfinite induction to prove that for any ordinal \( \beta \)

\( \alpha < \beta \implies d_\alpha \leq d_\beta. \)

For limit ordinals \( \lambda \) we consider two cases: the zero and the non-zero. The zero case is trivial—the precondition is never satisfied; and in the non-zero case, for \( \alpha < \lambda \) the consequent of the implication boils down to the tautology

\( d_\alpha \leq \bigvee_{\beta < \lambda} d_\beta. \)

For successor ordinals \( \beta + 1 \) we also consider two cases: whether the \( \beta \) itself is a successor or not. If \( \beta = \alpha + 1 \), then by the induction hypothesis and by monotonicity of \( f \):

\[
d_\alpha \leq d_{\alpha+1} = d_\beta \implies f \ d_\alpha \leq f \ d_\beta \text{ iff } d_\beta \leq d_{\beta+1}
\]

and the result follows for arbitrary \( \alpha < \beta + 1 \) by transitivity. If \( \beta \) is a limit ordinal, then for \( \beta = 0 \) the result follows trivially as \( d_0 = \bot \leq d_{\beta+1} \); for other limit ordinals \( \beta = \lambda \), first,

\( \alpha < \lambda \implies d_\alpha \leq \bigvee_{\alpha < \lambda} d_\alpha \)

\( \implies f \ d_\alpha \leq f \left( \bigvee_{\alpha < \lambda} d_\alpha \right) \)

5 A related, classical, result was given by Bourbaki [1949] and Witt [1950]: In a pointed partial order that is closed under taking least upper bounds, every inflationary map \( f \)—i.e. for which every \( x \) in the domain of \( f \) satisfies \( x \leq f \ x \)—has a fix-point.
STRONG NORMALIZATION

iff $d_{\alpha+1} \leq d_{\lambda+1}$;

and applying the induction hypothesis, to $\alpha < \alpha + 1$ we get

$$\implies d_\alpha \leq d_{\alpha+1} \leq d_{\beta+1};$$

by the least upper bound property $d_\beta$, we find that $d_\beta \leq d_{\beta+1}$.

**Theorem 34.** For any monotone $f : \text{ONP} \to \text{ONP}$, the chain $d_\beta$ defined in proposition 16 has a fix-point.

**Proof.** Take the set

$$S = \{ d_\beta \mid \beta \in \text{Ord} \} \subseteq \text{ONP}$$

We’ve already proven that $d_\beta$ was an increasing chain; the non-existence of a fix-point would mean that the chain is strictly increasing and that the mapping

$$\beta \mapsto d_\beta$$

is an injection from the set of ordinals to the set $S$. However this contradicts the following standard result, attributed to Hartogs (1915), on ordinals:

*For every set $S$ there is a least ordinal $\alpha$ such that there is no injection from $\alpha$ to $S$.*

Classically, then, there must be a least $\beta$ such that $d_\beta = d_{\beta+1} = f d_\beta$.

---

§6.22. Properties of the Fix-point

This fix-point is, in fact, the least; to prove this formally would require an additional transfinite induction. We shan’t engage in that proof, though; the properties that we need to reason inductively about the fix-point of a monotone extension of a function (downward closure) also allow us to establish the properties we need for the least fix-point.

§6.23. Admissibility

To any set, or property, $P$ satisfying

1. **Lub Preservation:** $S \subseteq P \implies P(\bigvee S)$

2. **Downward Closure:** $a \leq b$ and $P(b) \implies P(a)$

we shall call *admissible*. Note that condition 1 devolves in the empty case to $\bot \in P$.

The starting point for this definition is that of a Scott admissible set—a set which contains the least element and is closed under taking least upper bounds. However, because we are interested in monotone extensions of functions (for induction), we must strengthen our conditions to account for the least upper bound taken (hence lub preservation) and its domain (hence downward closure).

---

6For historical details on this terminology, see the book by Winskel (1993, notes at the end of chapter 10).

7Note that the two admissibility conditions imply the conditions for an ideal set: downward closure and the existence in the ideal of an upper bound of any pair of elements in the ideal.
Theorem 35 (Induction Principle for Monotone Extensions of Endofunctions). Take \( L \) to be a complete lattice; let \( f : L \to L \) be an endofunction (not necessarily a homomorphism) and \( P \) be an admissible property on (read, subset of) \( L \); if
\[
P(a) \implies P(f(a)),
\]
then property \( P \) holds for the least fix-point of its monotone extension,
\[
P(\text{lfp}(\lceil f \rceil))
\]

Proof. The monotone extension of any \( f : L \to L, \lceil f \rceil \), is monotone. It follows from theorem 34 that, for some ordinal \( \phi \), the following function, defined by transfinite recursion, has a fix-point:
\[
\begin{align*}
d_{\beta+1} &= \lceil f \rceil d_{\beta} & \text{for successor ordinals} \\
d_\lambda &= \bigvee_{\beta < \lambda} d_{\beta} & \text{for limit ordinals}
\end{align*}
\]
It suffices, to prove that for every \( a \in \text{Ord} \), we have that \( P(d_a) \); for it follows then that \( P(d_a) \) holds for the fix-point \( d_\phi \). As we necessarily have that \( \text{lfp}(\lceil f \rceil) \leq d_\phi \), by the downward closure of \( P \) we conclude
\[
P(\text{lfp}(\lceil f \rceil)).
\]
The proof proceeds by transfinite recursion on ordinals assuming that for a given ordinal \( \beta \) the property \( P(d_\beta) \) holds for all ordinals \( a < \beta \) strictly below. Limit ordinals \( \lambda \) can be quickly disposed with: using the induction hypothesis, define the set
\[
S = \{ d_a \mid a < \lambda \} \subseteq P
\]
whence, by admissibility (condition 1), it follows that
\[
P\left( \bigvee_{a < \lambda} d_a \right).
\]
For the successor case, \( \alpha + 1 \), we assume \( P(d_a) \) holds. It follows by the downward closure of \( P \) that for any \( a \leq d_a \) in \( L \), \( P(a) \) also holds. In set form this is just
\[
\{ a \mid a \leq d_a \} \subseteq P \quad \text{whence by the assumption on } f \quad \{ f(a) \mid a \leq d_a \} \subseteq P.
\]
This set satisfies condition 1 of an admissible set, and, therefore, we have that
\[
P\left( \bigvee_{a \leq d_a} f(a) \right) \quad \text{or, equivalently,} \quad P(\lceil f \rceil d_a)
\]
which is nothing more than \( P(d_{\alpha+1}) \). 
\[\square\]
§6.24. And gfps? One might wonder why not do the same for gfps, seen as they are essential for co-induction. Since we work with complete—self-dual—lattices, we can reuse the above theorem to give us the intended co-induction principle.

Corollary 7. Let \( g : M \to M \) be an endofunction (not necessarily a homomorphism) and \( P \) be a property (alternatively read subset) on a complete lattice \( M \) for which:

1. Glb Preservation: \( S \subseteq P \implies P(\bigwedge S) \)
2. Upward Closure: \( a \leq b \) and \( P(a) \implies P(b) \)
3. Function Preservation: \( P(a) \implies P(g \cdot a) \)

The property holds for the greatest fix-point of its monotone restriction,

\[
P(gfp([g])).
\]

Proof. Apply theorem \([35]\) for \( L = M^* \).

Lemma 33 (Conservation). For any typescheme \( F \), type \( A \), and interpretation context \( \gamma \) suitable for both,

\[
\min \left\{ (\mu F(X).F(X))(\gamma)^T \right\} \subseteq (\mu X.F(X))(\gamma)^T
\]

\[
motr_{r,a} \left( \bigcap_{N \in \mathcal{L} \cap N} (\mu F(X)(\gamma[X \mapsto N])^T) \right) \subseteq (\mu X.F(X))(\gamma)^T
\]

\[
mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{L} \cap N} (\nu X.F(X)(\gamma[X \mapsto N])^T) \right) \subseteq (\nu X.F(X))(\gamma)^T
\]

\[
mout [ (\mu F(X))(\gamma)^T ] \subseteq (\nu X.F(X))(\gamma)^T
\]

where \( X \) is fresh for type \( A \).

Proof. Inductive Constructor: Take some \( \min(t) \in \min \left\{ (\mu F(X).F(X))(\gamma)^T \right\} \); it follows, via substitution, that

\[
t \in (\mu F(X).F(X))(\gamma)^T = (\mu X.F(X)(\gamma[X \mapsto [\mu X.F(X)](\gamma)]))
\]

From here, the calculations go as follows:

\[
\min(t) \in \min \left\{ (\mu F(X)(\gamma[X \mapsto [\mu X.F(X)](\gamma)]))^T \right\}
\]

\[
= \min \left\{ (-)^T \cdot (\mu F(X)(\gamma[X \mapsto [\mu X.F(X)](\gamma)])) \right\}
\]

\[
= \left( \min \left\{ (-)^T \cdot (\mu F(X)(\gamma[X \mapsto -])) \right\} \right) \cdot (\mu X.F(X)(\gamma))
\]

\[
\subseteq \bigcup_{N \leq [\mu X.F(X)](\gamma)} (\min \left\{ (-)^T \cdot (\mu F(X)(\gamma[X \mapsto -])) \right\} \cdot N)
\]

\[
= \bigcup_{N \leq [\mu X.F(X)](\gamma)} (\mu F(X)(\gamma[X \mapsto -]))^T
\]

157
REDUCTION PROPERTIES OF THE EXTENSION TO MENDLER-INDUCTIVE TYPES

\[ = \left( \bigvee_{N \leq \mu X. F(X)} (\mu P \circ \{ X \mapsto - \}) \right) \]

\[ = \left( [\mu P \circ \{ X \mapsto - \}] [\mu X. F(X)] \right) \]

The function we are applying the inductive type to is exactly that of the least fix-point definition and, by the latter’s defining property, we get our intended

\[ \leq (\mu X. F(X)) \]

(lemma 30)

Inductor:

The challenge for the eliminator is to prove that the statement of conservation can be construed as a proposition within the confines of our induction principle (theorem 35). Using the—by now familiar—fact that the terms (and co-terms) in the normal interpretation are included in the orthogonal one, and defining the abbreviation

\[ \exists \mu X. F(X) \]

our goal can be re-framed as

\[ \exists \mu X. F(X) \]

with the \( ONP \) interpretation of the inductive type being the least fixed point

\[ \text{lfp}([\mu P \circ \{ X \mapsto - \}]). \]

Take then \( P \) to be

\[ P(N) \]

iff \( \exists \mu X. F(X) \]

To use the induction principle (theorem 35) we need to prove that \( P \) is adequate (downward and least upper bound closed) and that it is preserved by \( \mu P \circ \{ X \mapsto - \} \).

Let \( M \leq N \in ONP \). For downward closure, by contravariance of the order for the continuation side, it follows that \( (N)^K \leq (M)^K \); whence, if \( P(N) \) holds, we have that

\[ \exists \mu X. F(X) \]

or, equivalently, \( P(M) \), as needed. For the least upper bound property, for \( S \subseteq P \) we consider the empty and non-empty cases separately. If \( S = \emptyset \) then, trivially,

\[ \exists \mu X. F(X) \]

158
otherwise,

for any \( N \in S \), \( \text{mitr}_{\rho,a} \left[ \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^K \right] \subseteq (N)^K \)

iff \( \text{mitr}_{\rho,a} \left[ \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^K \right] \subseteq \bigcap_{N \in S} (N)^K \)

iff \( \text{mitr}_{\rho,a} \left[ \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^K \right] \subseteq \left( \bigvee_{N \in S} N \right)^K \)

iff \( \mathcal{P}\left( \bigvee S \right) \)

The core of the proof lies on showing preservation of \( \mathcal{P} \) by \( \text{MuP} \circ \langle F(X) \rangle(\gamma[X \mapsto -]) \); assume, to this end, that \( N \in \mathcal{OP} \) is such that \( P(N) \), from whence we gather that

\[
\text{mitr}_{\rho,a} \left[ \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^K \right] \subseteq (N)^K
\]

iff \( \text{mitr}_{\rho,a} \left[ \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^K \right] \subseteq \left( \bigvee X \langle \gamma[X \mapsto N] \rangle \right)^K \)

iff \( \text{mitr}_{\rho,a} \left[ \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^K \right] \subseteq (\langle X \rangle(\gamma[X \mapsto N]))^K \)

iff \( \mu_a.(\text{mitr}_{\rho,a} \left[ \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^K \right] /a) \subseteq (\langle X \rangle(\gamma[X \mapsto N]))^K \)

iff \( \mu_a.(\text{mitr}_{\rho,a} \left[ \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^K \right] /a) \subseteq \left( X - A \right)^K(\gamma[X \mapsto N])^K \)

iff \( \mu_a.(\text{mitr}_{\rho,a} \left[ \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^K \right] /a) \subseteq \left( X - A \right)^K(\gamma[X \mapsto N])^K \)

This will put us in a position to prove that the inductive calls that comprise the left-hand side of the inclusion satisfy the inductive restriction. To see this, take

\[
\text{mitr}_{\rho,a}[k, l] \in \text{mitr}_{\rho,a} \left[ \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^K \right]
\]

from which we have that \( k \in \bigcap_{N \in \mathcal{NP}} F(N) \). Specifically for our assumed \( N \) that satisfies property \( \mathcal{P} \)

\[
k \in (\langle F(X) \rangle(\gamma[X \mapsto N]))^K(\langle X - A \rangle(\gamma(X \mapsto N)))/a;
\]

and, from *,

\[
\mu_a.(\text{mitr}_{\rho,a}[k, a]) \in (\langle X - A \rangle(\gamma[X \mapsto N]))^K;
\]

combining these two observations with the definition of the substitution restriction yields

for any \( l' \in (\langle A \rangle(\gamma))^K \), \( k[\mu_a.(\text{mitr}_{\rho,a}[k, a]) /a][l'/a] \in (\langle F(X) \rangle(\gamma[X \mapsto N]))^K. \)

As claimed, then,

\[
k \in (\langle F(X) \rangle(\gamma[X \mapsto N]))^K/a(\langle A \rangle(\gamma))^K.
\]
From this we conclude that
\[
\text{mitr}_{\rho,\alpha} \left[ \bigcap_{N \in \mathcal{N}P} F(N), (\langle A \rangle(\gamma))^K \right] \subseteq \text{mitr}_{\rho,\alpha} \left[ (\langle F(X) \rangle(\gamma[X \mapsto N]))^K / \rho(\langle A \rangle(\gamma))^K, (\langle A \rangle(\gamma))^K \right]
\]
\[
\subseteq \bigcup_{Q \in \mathcal{N}P} \text{mitr}_{\rho,\alpha} \left[ (\langle F(X) \rangle(\gamma[X \mapsto N]))^K / \rho(Q)^K, (Q)^K \right]
\]
\[
= (\text{MuP}(\langle F(X) \rangle(\gamma[X \mapsto N])))^K
\]
\[
= ((\text{MuP} \circ \langle F(X) \rangle(\gamma[X \mapsto -])) N)^K
\]

which is nothing but \( P((\text{MuP} \circ \langle F(X) \rangle(\gamma[X \mapsto -])) N) \).

**Co-inductor:** Similarly, for the co-inductor, it suffices (lemma 30) to prove that
\[
\text{mcoitr}_{r,x} \left[ \bigcap_{N \in \mathcal{N}P} (\langle F(X) \rangle(\gamma[X \mapsto N]))^T, (\langle A \rangle(\gamma))^T \right]
\]
\[
\subseteq (\lceil vX. F(X) \rceil(\gamma))^T;
\]
the \( \mathcal{N}P \) interpretation of a co-inductive type \( vX. F(X) \) is given by
\[
gfp([\text{NuP} \circ \langle F(X) \rangle(\gamma[X \mapsto -])]).
\]

Define the abbreviation
\[
F(N) \equiv (\langle F(X) \rangle(\gamma[X \mapsto N]))^T / (\langle A \rangle(\gamma))^T / r.
\]

Our goal then is to prove that
\[
\text{mcoitr}_{r,x} \left[ \bigcap_{N \in \mathcal{N}P} F(N), (\langle A \rangle(\gamma))^T \right] \subseteq (\text{gfp}([\text{NuP} \circ \langle F(X) \rangle(\gamma[X \mapsto -])]))^T
\]

which, we will show, falls within the scope of corollary 7, the induction principle for greatest fix-points. Concretely, the property \( P \) that we’re trying to ascribe to the greatest fix-point is
\[
P(N) \quad \text{iff} \quad \text{mcoitr}_{r,x} \left[ \bigcap_{N \in \mathcal{N}P} F(N), (\langle A \rangle(\gamma))^T \right] \subseteq (N)^T.
\]

As usual, we separate the verification of the meet properties between the empty and the non-empty cases. For the empty case we have
\[
\text{mcoitr}_{r,x} \left[ \bigcap_{N \in \mathcal{N}P} F(N), (\langle A \rangle(\gamma))^T \right] \subseteq SNT = (T)^T;
\]
for the non-empty case, take $\emptyset \subset S \subseteq P$ and reason

$$\begin{align*}
\text{for any } N \in S, \ mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^T \right) &\subseteq (N)^T \\
\text{iff } mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^T \right) &\subseteq \bigcap_{N \in S} (N)^T \\
\text{iff } mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^T \right) &\subseteq \left( \bigcap_{N \in S} N \right)^T \\
\text{iff } P\left( \bigwedge S \right)
\end{align*}$$

Upward-closure is a direct consequence of the definition of the order. Take two orthogonal normal pairs $N \leq M$, and reason

$$\begin{align*}
P(N) \text{ iff } mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^T \right) &\subseteq (N)^T \\
\Rightarrow mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^T \right) &\subseteq (M)^T \\
\text{iff } P(M).
\end{align*}$$

All that remains to prove is the preservation of the property by the interpretation of typeschemes $F(X)$. Assume then that $P(N)$ holds for some arbitrary $N \in \mathcal{NP}$; our first goal is to show that the substitution with the co-inductive step is valid:

$$\begin{align*}
mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^T \right) &\subseteq (N)^T \\
\text{iff } mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^T \right) &\subseteq ([X](\gamma[X \mapsto N]))^T \\
\Rightarrow mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^T \right) &\subseteq ([X]^T(\gamma[X \mapsto N]))^T \\
\text{iff } mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), x \right) &\subseteq (\langle X \rangle(\gamma[X \mapsto N]))^{(\langle A \rangle(\gamma))^T} \\
\text{iff } mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), x \right) &\subseteq (\langle X \rangle(\gamma[X \mapsto N]))^{(\langle A \rangle(\gamma[X \mapsto N]))^T} \\
\text{iff } \lambda x. (mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), x \right)) &\subseteq \lambda x. ([X]^T(\gamma[X \mapsto N]))^{(\langle A \rangle(\gamma[X \mapsto N]))^T} \\
\Rightarrow \lambda x. (mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), x \right)) &\subseteq ([A \mapsto X]^T(\gamma[X \mapsto N]))^{(\langle A \rangle(\gamma[X \mapsto N]))^T} \\
\Rightarrow \lambda x. (mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), x \right)) &\subseteq ([A \mapsto X]^T(\gamma[X \mapsto N]))^{(\langle A \rangle(\gamma[X \mapsto N]))^T}
\end{align*}$$

Then, pointwise, we get that for any

$$mcoitr_{r,x}(t, u) \in mcoitr_{r,x} \left( \bigcap_{N \in \mathcal{NP}} F(N), (\langle A \rangle(\gamma))^T \right)$$

we have, simultaneously,

$$t \in \bigcap_{N \in \mathcal{NP}} F(N) \text{ and } \lambda x. (mcoitr_{r,x}(t, x)) \in ([A \mapsto X]^T(\gamma[X \mapsto N]))^{(\langle A \rangle(\gamma[X \mapsto N]))^T};$$
REDUCTION PROPERTIES OF THE EXTENSION TO MENDLER-INDUCTIVE TYPES

in particular, for the $N$ we are working with,

$$t \in (\langle F(X) \rangle(\gamma[X \mapsto N]))^T / \langle A \rangle^T / x$$

and

$$\lambda x. (mcoitr_{r,s}(t,x)) \in (\langle A \mapsto X \rangle(\gamma[X \mapsto N]))^T;$$

and from this it follows that,

for any $u' \in (\langle A \rangle^T)$, $t[\lambda x. (mcoitr_{r,s}(t,x)) / r] [u' / x] \in (\langle F(X) \rangle(\gamma[X \mapsto N]))^T$;

whence,

$$t \in (\langle F(X) \rangle(\gamma[X \mapsto N]))^T / x (\langle A \rangle^T)$.

From the above it follows that

$$mcoitr_{r,s} \left( \bigcap_{N \in \Omega \cap P} P(N), (\langle A \rangle^T) \right)$$

$$\subseteq mcoitr_{r,s} \left( \langle F(X) \rangle(\gamma[X \mapsto N])^T / x (\langle A \rangle^T), (\langle A \rangle^T) \right)$$

$$\subseteq \bigcup_{\Omega \cap P \neq \varnothing} mcoitr_{r,s} \left( \langle F(X) \rangle(\gamma[X \mapsto N])^T / x (Q)^T, (Q)^T \right)$$

$$= (NuP(\langle F(X) \rangle(\gamma[X \mapsto N])))^T$$

$$= ((NuP \circ (\langle F(X) \rangle(\gamma[X \mapsto -]))) N)^T$$

which is exactly our intended

$$P((NuP \circ (\langle F(X) \rangle(\gamma[X \mapsto -]))) N).$$

Co-inductive Eliminator: For the co-injection, we can just use the substitution lemma followed by the fix-point property:

$$mout[\langle (\langle F(vX.F(X)) \rangle(\gamma))^T \rangle]$$

$$= mout [\langle (\langle F(X) \rangle(\gamma[X \mapsto vX.F(X)](\gamma)))^K \rangle]$$

$$\subseteq (NuP(\langle F(X) \rangle(\gamma[X \mapsto vX.F(X)](\gamma))))^K$$

$$= (NuP \circ (\langle F(X) \rangle(\gamma[X \mapsto -])), [vX.F(X)](\gamma))^K$$

$$\subseteq \bigcup_{N \leq [vX.F(X)](\gamma)} (NuP \circ (\langle F(X) \rangle(\gamma[X \mapsto -]) N)^K$$

$$= \left( \bigcap_{N \leq [vX.F(X)](\gamma)} (NuP \circ (\langle F(X) \rangle(\gamma[X \mapsto -]) N \right)^{\gamma K}$$

$$= (([NuP \circ (\langle F(X) \rangle(\gamma[X \mapsto -]) [vX.F(X)](\gamma)]^K$$

$$= (([NuP \circ (\langle F(X) \rangle(\gamma[X \mapsto -]) \text{ gfp}([NuP \circ (\langle F(X) \rangle(\gamma[X \mapsto -]) ]))^K$$

162
6.5.5 Strong Normalization for Inductive Types

**Theorem 36 (Adequacy).** Let \( t \) be a term, \( k \) a co-term, and \( c \) a cut of the second-order Dual Calculus extended with Mendler induction; for any contexts \( \Gamma \) and \( \Delta \), and types \( T \) for which they are well-typed,

\[
\Gamma \vdash t : T \mid \Delta, \quad \Gamma \vdash k : T \vdash \Delta, \quad \Gamma \vdash c \vdash \Delta,
\]

any interpretation context \( \gamma \) and substitution \( \sigma \) for \( \Gamma \) and \( \Delta \) validate

\[
t[\sigma] \in (\langle T \rangle(\gamma))^T, \quad k[\sigma] \in (\langle T \rangle(\gamma))^K, \quad c[\sigma] \in SN.
\]

**Proof.** We extend the proof of theorem [14], here, with only the inductive and co-inductive cases.

**Inductive Types:** Beginning with the injection into inductive types, we get, straightforwardly from the induction hypothesis and conservation,

\[
t[\sigma] \in (\langle \mu X. F(\mu X. F) \rangle(\gamma))^T \quad \text{(IH)}
\]

\[
\implies \min(t[\sigma]) \in (\langle \mu X. F(\mu X. F) \rangle(\gamma))^T
\]

\[
\implies \min(t)[\sigma] \in (\langle \mu X. F(\mu X. F) \rangle(\gamma))^T
\]

For the iterator, the proof almost exactly boils down to proving that the induction hypothesis on \( k[\sigma] \) (and \( l[\sigma] \)) implies the relevant pre-conditions of conservation (lemma 33). A slight complication arises from the fact that the free type variables in \( A \) need not appear in the conclusion, and therefore a context \( \gamma \) which satisfies the adequacy conditions for the conclusion, need not be suitable for the antecedents. To guarantee that that is the case we extend any such \( \gamma \) with those type variables that appear in \( A \) but were not contemplated in \( \gamma \); denoting the set of these by \( C = \text{ftv}(A) - \text{dom}(\gamma) \) we get a new context by assigning \( \bot \) to those variables:

\[
\gamma' = \gamma[C \mapsto \bot].
\]

We shall also need to consider a further extension that accounts for the extra (fresh) type variable \( X \); this freshness can be used—by the weakening property—to assume that we assign an arbitrary \( N \in \mathcal{ONP} \) to it. The set \( C \) is necessarily finite as it is bound by the free type variables that appear in the (finite) type \( A \). By repeated applications of weakening we

\footnote{Indeed any arbitrary but fix element of \( \mathcal{ONP} \) would do.}
have that any substitution $\sigma$ in the adequacy conditions for the conclusion w.r.t. context $\gamma$ is also valid for $\gamma'$ as

$$x : T \in \Gamma \implies \sigma(x) \in (\langle T \rangle(\gamma'))^T = (\langle T \rangle(\gamma'[X \mapsto N]))^T,$$

$$a : T \in \Delta \implies \sigma(a) \in (\langle T \rangle(\gamma'))^K = (\langle T \rangle(\gamma'[X \mapsto N]))^K.$$

For the return continuation $l$ we immediately get from the induction hypothesis and weakening that

$$l[\sigma] \in \langle A \rangle(\gamma') = \langle A \rangle(\gamma'[X \mapsto N]).$$

For the inductive step $k$, for fresh $\rho \neq \alpha \in \text{Covar}$, and any

$$k' \in \langle X - A \rangle(\gamma'[X \mapsto N]) \quad \text{and} \quad l' \in \langle A \rangle(\gamma') = \langle A \rangle(\gamma'[X \mapsto N])$$

we have, then, that $\sigma[m/\rho][l'/a]$ is a substitution for $k$ in the conditions of the theorem for $\gamma'[X \mapsto N]$; therefore

$$k[\sigma][m/\rho][l'/a] = k[\sigma[m/\rho][l'/a]] \in (\langle F(X) \rangle(\gamma'[X \mapsto N]))^K.$$

By the definition of the restriction, and because the $N$ above was chosen arbitrary,

$$k[\sigma] \in \bigcap_{N \in \text{ONT}} (\langle F(X) \rangle(\gamma'[X \mapsto N]))^K_{\langle(\langle X - A \rangle(\gamma'[X \mapsto N]))^\rho/l'\rangle_{\langle\langle A \rangle(\gamma')^\rho/a\rangle}}.$$

The conservation lemma then yields

$$\text{mitr}_{\rho,a}[k,l][\sigma] = \text{mitr}_{\rho,a}[k[\sigma],l[\sigma]] \in (\langle \mu X . F(X) \rangle(\gamma'))^K,$$

which, by weakening (ftv($\mu X . F(X)$) $\subseteq$ dom($\gamma$)), is equivalent to

$$\text{mitr}_{\rho,a}[k,l][\sigma] = \text{mitr}_{\rho,a}[k[\sigma],l[\sigma]] \in (\langle \mu X . F(X) \rangle(\gamma'))^K.$$

**Co-inductive Types:** The proof complexity is reversed on co-inductive types: terms are complicated; continuations are not. For the former, take a judgement

$$\Gamma \vdash \text{mcoitr}_{r,x}(t,u) : \nu X . F(X) \upharpoonright \Delta;$$

the rule hypothesis on the co-iteration step, $x : A, r : A \rightarrow X, \Gamma \vdash t : F(X) \upharpoonright \Delta$, has the type variable $X$ and the free type variables in $A$ (assumed different from $X$) free in addition to those in the conclusion. We also have term variables $r$ and $x$ which are not present in the conclusion. To apply the induction hypothesis to $t$, we need to provide suitable values for those parameters. As we did for cuts in theorem [14], for any context $\gamma$ in the conditions of adequacy for the conclusion, we begin by creating a new context $\gamma'$ that covers the extra type variables in $A$—denoted by the set $C = \text{ftv}(A) - \text{dom}(\gamma)$—by setting their interpretation to $\bot$,

$$\gamma' = \gamma \upharpoonright C \mapsto \bot.$$
STRONG NORMALIZATION

For any \( N \in \mathcal{N} \), then, the context \( \gamma'[X \mapsto N] \) is in the conditions of the induction hypothesis. For any \( x : T \in \Gamma \), we have that \( \text{frv}(T) \cap C = \emptyset \) so that by (finite applications of) weakening,

\[
\langle T \rangle(\gamma) = \langle T \rangle(\gamma').
\]

Any substitution \( \sigma \) that was in the adequacy conditions for \( \gamma, \Gamma, \) and \( \Delta \) is therefore also in the conditions for \( \gamma'[X \mapsto N], \Gamma, \) and \( \Delta \). Extending this \( \sigma \) with arbitrary \( t' \in (\langle A \rightarrow X \rangle (\gamma'[X \mapsto N]))^T \) and \( u' \in (\langle A \rangle (\gamma'))^T = (\langle A \rangle (\gamma'[X \mapsto N]))^T \)

yields a substitution in the conditions of the theorem for \( t \). We can then apply the induction hypothesis to yield

\[
t[\sigma][t'/r][u'/x] = t[\sigma][t'/r][u'/x] \in (\langle F(X) \rangle (\gamma'[X \mapsto N]))^T.
\]

As \( N, t', \) and \( u' \) were chosen arbitrary and by the definition of restriction under substitution, we have that

\[
t[\sigma] \in \bigcap_{N \in \mathcal{N} \cap \mathcal{P}} (\langle F(X) \rangle (\gamma'[X \mapsto N]))^T/{(\langle A \rangle (\gamma'[X \mapsto N]))^T/r}.
\]

As for term \( u \), the contexts and types in the antecedent of the rule are \( \Gamma, \Delta, \) and \( A; \) therefore, the context \( \gamma' \) and the substitution \( \sigma \) considered above satisfy the requirements of the adequacy condition, and the induction hypothesis asserts that

\[
u[\sigma] \in (\langle A \rangle (\gamma'))^T = (\langle A \rangle (\gamma'[X \mapsto N]))^T.
\]

Applying the conservation lemma, we get that

\[
mcoitr_{r,x}(t[\sigma], u[\sigma]) \in (\langle \nu X. F(X) \rangle (\gamma'))^T.
\]

Applying weakening repeatedly to the interpretation—as the free type variables in the inductive type are assumed to be covered by \( \gamma \)—results in the desired

\[
mcoitr_{r,x}(t,u)[\sigma] \in (\langle \nu X. F(X) \rangle (\gamma))^T.
\]

On the continuation side, we have that the free type and term variables do not change from hypothesis to conclusion. Applying the induction hypothesis to the co-term on the antecedent of the rule yields

\[
k[\sigma] \in (\langle F(\nu X. F(X)) \rangle (\gamma))^K
\]

\[
mout[k][\sigma] \in (\langle \nu X. F(X) \rangle (\gamma))^K \quad \text{(Conservation)}
\]

\[
mout[k][\sigma] \in (\langle \nu X. F(X) \rangle (\gamma))^K
\]

\[\square\]

Corollary 8 (Strong Normalization). Every well-typed phrase of DC with Mendler induction is strongly normalizing.
REDUCTION PROPERTIES OF THE EXTENSION TO MENDLER-INDUCTIVE TYPES
Chapter 7

Concluding Remarks

7.1 Computed So Far

§7.1. A General Logic Begets a General Programming Language  The central tenet of this thesis, declared from the very onset, was that, making use of the Curry-Howard isomorphism, one could come to interesting Computational structure starting from Logic, rather than getting there by appealing to any a priori intuition. So, we chose a very general Logic—in the sense that it validates a maximal number of theorems without sacrificing consistency—and translated it into a computational calculus which turned out to be powerful enough to include both the functional and the imperative (read, control) worlds. The system thus created was not only very general but also well-behaved.

§7.2. Further Generality  The above being said, it would be silly to reject computational intuitions altogether. In particular, induction is a central theme in Theoretical Computer Science, and the obvious question is how this could be include within the above system. But chapter 4 showed how to—again, appealing to Logical intuition—tame induction, reaching at Mendler induction.

§7.3. Reaping the Rewards of Generality  The next step—and the original technical contribution of this thesis—was then to marry Classical Logic with Mendler induction (chapter 5) in a way that was provably well-behaved: the resulting calculus was subject preserving, substitutive, and strongly-normalizing (chapter 6).

We also showed how Intuitionistic Mendler induction and ordinary Classical induction were subsumed by this system. In the possession of these embeddings a host of important results follow as corollaries of the work presented herein:

Corollary 9. The simply-typed lambda-calculus (Church, 1940) is strongly-normalizing.

Corollary 10. System F (Girard, 1972; Reynolds, 1974) is strongly normalizing.


Corollary 12. The simply-typed lambda-calculus with ordinary induction is strongly normalizing.

Corollary 13. System F with ordinary induction is strongly normalizing.

7.2 Some Possible Continuations

§7.4. Categorically Mute Having concluded a purely operational study of the Dual Calculus, the obvious next step is to take this study into the denotational—and even more, categorical—level. Categorical models of Classical Logics are known to very easily collapse into pre-orders (see, e.g., Lambe and Scott, 1988; Selinger, 2001); and as was observed elsewhere (Bellin et al., 2006) most alternatives that are not pre-orders are either just double-negation translations (Selinger, 2001) or “parasitic on experience with Linear Logic” (Führmann and Pym, 2007). However, there is a categorical distillation of (functional) Mendler induction due to Uustalu and Vene (1999); can this idea be extended to those categorical models of Classical Logic, or to other models of a more polycategorical inclination (Szabo, 1975; Bellin et al., 2006)?

§7.5. DC, the Intermediate Language Looking at DC beyond pure programming, there is the obvious—and aforementioned—connection between the language and continuation semantics. Continuations have a conspicuous presence in the world of compilers where they can be used as an intermediary representation of a program (e.g., Danvy and Filinski, 1992; Sabry and Felleisen, 1993). From this translation, in which individual instructions affect the global representation of a program, the compiler can statically merge many of the intermediate stages of reduction. The image of any CPS transform however tends to be composed of unwieldy looking terms, hampering their understanding; by comparison, DC phrases are neater and should clarify any usable structure within these translations.

§7.6. DC, the Realizer Preliminary investigations indicate that Krivine’s system of realizability for Classical Logic (Krivine, 2009; Streicher, 2013) is easily encodable within DC. It would be interesting to revisit the system in the context of the study we have produce herein. Specifically, Krivine (2009) showed how to use Church-encodings to specify realizers for assertions on natural numbers. But this suggests using a proper inductive principle—even better, a very general inductive principle—as the generator for realizers.

§7.7. DC, the Meta-language The final excursion we mention takes us full circle re the Curry-Howard isomorphism, back to Logical concerns. As a community, we usually define our programming languages (typings included) by some sort of implicit induction on the “syntax formation rules”. But it is often the case that this does not immediately create the necessary structure one might need to analyse the language: the proof of certain results—like strong normalization—requires the definition of the contexts that may enclose a given term. E.g., Pitts and Stark (1998) start from a functional language with local state, and from there determine an auxiliary calculus where terms are paired up with their continuations. They then show by structural induction the strong normalization of this auxiliary calculus and that this, in turn, implies the strong normalization of the initial language. We posit that the Dual Calculus with its explicit values and continuations can serve as a meta-language for this sort of proofs. If so, the extension to Mendler inductive
SOME POSSIBLE CONTINUATIONS

types that has been the subject of our journey should lead to an incredibly powerful proof discipline.
CONCLUDING REMARKS
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


J. Lambek and P. Scott. Introduction to higher order categorical logic, volume 7. Cambridge Univ Pr, 1988.


BIBLIOGRAPHY


BIBLIOGRAPHY


