Analysis of methods for extraction of programs from non-constructive proofs

Trifon Trifonov

Dissertation
an der Fakultät für Mathematik, Informatik, Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Trifon Trifonov
18 August 2011

Trifon Trifonov

Analysis of methods for extraction of programs from non-constructive proofs

Dissertation an der Fakultät für
Mathematik, Informatik und Statistik der
Ludwig-Maximilians-Universität München

1. Berichterstatter: Prof. Dr. Helmut Schwichtenberg
2. Berichterstatter: Prof. Dr. Wilfried Buchholz
3. Prüfer: Prof. László Erdős, Ph.D.
4. Prüfer: Prof. Dr. Otto Forster
Ersatzprüfer: Prof. Dr. Franz Merkl
Externer Gutachter: Prof. Dr. Thierry Coquand

Vorgelegt im 18 August 2011
Tag der Disputation: 15 Februar 2012

Abstract

Proofs in constructive logic correspond to functional programs in a direct and natural way. Computational content can also be found in proofs which use non-constructive principles, but more advanced techniques are required to interpret such proofs. Various methods have been developed to harvest programs from derivations in classical logic and experiments have yielded surprising and counterintuitive, yet correct and efficient algorithms. Nevertheless, the use of non-constructive arguments generally leads to an indirect backtracking computation, which is slower and more difficult to understand as compared to a program extracted from a constructive proof.

Constructive proofs can be transformed into programs in an unambiguous manner by projecting only their computational components. However, for proofs in classical logic there seems to be no canonical definition of their computational meaning, since every extraction method introduces specific computational infrastructure to support non-constructive reasoning. Applying several techniques to the same proof can result in different correct programs, however the relation between them with respect to efficiency, size and readability has not been thoroughly explored.

The first part of the present work compares two computational interpretations of non-constructive proofs: refined A-translation [BBS02] and Gödel's functional "Dialectica" interpretation [Göd58]. An arithmetical system is defined in which both techniques can be applied to the same proof object. The behaviour of the extraction methods is evaluated in the light of several case studies, and the resulting programs are compared. It is argued that the two interpretations correspond to specific backtracking schemes and that programs obtained via refined A-translation tend to be simpler, faster and more readable than programs obtained via Gödel's interpretation.

The second part of the thesis introduces three layers of optimisation of Gödel's interpretation to produce faster and more readable programs. First, it is shown that syntactic repetition of subterms can be reduced by using **let**-constructions instead of meta substitutions. The practical effects of the modification are nearly linear size of extracted terms and improved efficiency, achieved by avoiding repeated evaluations of equal terms. The second improvement is an extension of previous work [Ber05, Her07b], which allows declaring syntactically computational parts of the proof as computationally irrelevant. It is shown that Gödel's interpretation admits a wide variety of such annotations, which can be used to remove redundant parameters, possibly improving the efficiency of the program. An additional feature is the ability to embed Kreisel's modified realisability, and thus the refined A-translation, inside the extended Dialectica interpretation. Finally, a special case of induction is identified, for which a more efficient recursive extracted term can be defined. It is shown the outcome of case distinctions can be memoised by a boolean flag, which can result in exponential improvement of the average time complexity of the extracted program.

Zusammenfassung

Beweise in der konstruktiver Logik entsprechen funktionalen Programmen auf eine direkte und natürliche Art. Rechnerischer Inhalt kann auch in Beweisen gefunden werden, die nicht-konstruktiven Prinzipen anwenden, allerdings verlangen die Beweise fortgeschrittenere Techniken um interpretiert werden zu können. Verschiedene Verfahren wurden entwickelt, um Programme aus Beweisen in der klassischen Logik zu ermitteln und Experimente haben manchmal überraschende und nicht eingängige, jedoch korrekte und effiziente Algorithmen ergeben. Dennoch führt die Verwendung von nicht-konstruktiven Argumenten generell zu einer indirekten rückverfolgenden Berechnung, die langsamer und schwieriger nachzuvollziehen ist, im Vergleich zu einem Program, das von einem konstruktiven Beweis extrahiert ist.

Konstruktive Beweise lassen sich auf eindeutige Weise allein durch die Projektion ihrer rechnerischen Komponenten in Programmen umwandeln. Doch für Beweise in der klassischen Logik scheint es keine kanonische Definition ihrer rechnerischen Bedeutung zu geben, weil jede Extraktionsmethode eine spezifische rechnerische Infrastruktur erzeugt, um eine nicht-konstruktive Argumentation zu unterstützen. Die Anwendung verschiedener Techniken auf den gleichen Beweis kann zu verschiedenen korrekten Programmen führen, wobei ihre Beziehung zueinander hinsichtlich Effizienz, Größe und Lesbarkeit nicht gründlich erforscht ist.

Der erste Teil der vorliegenden Arbeit vergleicht zwei rechnerische Interpretationen von nicht-konstruktiven Beweisen: verfeinerte A-Übersetzung [BBS02] und Gödels funktionale “Dialectica” Interpretation [Göd58]. Ein arithmetisches System wird definiert, in dem beide Techniken auf dem gleichen Beweisobjekt angewendet werden können. Das Verhalten der Extraktionsmethoden wird am Beispiel mehrerer Fallstudien ausgewertet, in denen die resultierenden Programme analysiert und verglichen werden. Dabei wird argumentiert, dass die beiden Interpretationen bestimmten Implementierungen vom Rücksetzverfahren entsprechen und dass die über die verfeinerte A-Übersetzung erhaltenen Programme in der Regel einfacher, schneller und besser lesbar sind als Programme, die man durch Gödels Interpretation erhält.

Im zweiten Teil der Arbeit werden drei Optimierungsvarianten von Gödels Interpretation vorgestellt, um schnellere und besser lesbare Programme herzustellen. Erstens wird gezeigt, dass syntaktische Wiederholung von Teiltermen mit der Anwendung von **let**-Konstruktionen statt Meta-Substitutionen reduziert werden kann. Die praktischen Auswirkungen dieser Änderung sind fast linearer Größe und eine verbesserte Effizienz von extrahierten Terme durch Vermeidung von wiederholten Auswertungen der gleichen Bedingungen. Die zweite Verbesserung liegt in der Erweiterung von Ergebnissen in [Ber05, Her07b], die es ermöglichen, syntaktisch rechnerische Beweisteile als irrelevant oder rechnerisch einheitlich zu deklarieren. Weitergehend wird thematisiert, dass Gödels Interpretation über eine Vielzahl solcher Anmerkungen verfügt, die zur Entfernung von redundanten Parametern genutzt werden können, ggf. zur Verbesserung der Programmeffizienz. Eine weitere Besonderheit ist die Fähigkeit, Kreisels modifizierte Realisierbarkeit Interpretation und damit die verfeinerte A-Übersetzung in der erweiterten Dialectica Interpretation einzubinden. Abschließend wird ein Spezialfall der Induktion ermittelt, für welches ein effizienter rekursiv extrahierter Term definiert werden kann. Es wird gezeigt, dass ein Ergebnis der Fallunterscheidungen in einen booleschen Flag memoisiert werden kann, was zu einer exponentiellen Verbesserung der durchschnittlichen Zeitkomplexität des extrahierten Programms führen kann.

Acknowledgements

There are many people whose support made this work possible.

First of all, I would like to thank my supervisor Prof. Dr. Helmut Schwichtenberg, who welcomed me into the Munich logic group and greatly influenced my scientific development. With his noteworthy scientific rigour he provided me with irreplaceable guidance in the world of proof theory, which helped me to find and follow my own direction of research. I thank him for introducing me to the proof assistant Minlog: the formal reflection of his view on mathematical logic and constructive mathematics. I am grateful for his patience and understanding, which were very important to me.

I am very indebted to my friends and colleagues in the Munich logic group for their helpful advice, encouragement, suggestions and fruitful discussions on various mathematical and non-mathematical topics: Prof. Dr. Wilfried Buchholz, Basil Karadaís, Bogomil Kovachev, Diana Raţiu, Freiric Barrál, Josef Berger, Luca Chiarabini, Simon Huber, Stefan Schimanski. Many of the ideas were born during invaluable scientific discussions with colleagues with whom I had the pleasure of working with closely: Mircea-Dan Hernest, Monika Seisenberger, Paulo Oliva, Stefan Hetzl, Ulrich Berger. I am very grateful for their help.

For the financial support of my scientific work I gratefully acknowledge three different projects: MATHLOGAPS (MEST-CT-2004-504029), a Marie Curie Early Stage Training Site, which is responsible for attracting many young people to mathematical logic; the Bulgarian National Science Fund project DO 02-102/23.04.2009; and the European Social Fund project BG051PO001-3.3.04/28.08.2009.

My interest in mathematical logic and its applications was ignited by my teachers in the Faculty of Mathematics and Informatics at Sofia University "St. Kliment Ohridski". I would specifically like to thank Prof. Dr. Magdalina Todorova for introducing me to theoretical computer science; Prof. Dr. Ivan Soskov, who advised me to apply for the MATHLOGAPS scholarship; and Prof. Dr. Tinko Tinchev for his enormous work and enthusiasm in providing financial support to me and many other Bulgarian students. I would also like to thank my colleagues in the chairs of Computer Informatics and Mathematical Logic, and the Laboratory for Interactive Multimedia at the Sofia University for their overall support. I thank my friend Kalin Georgiev, with whom I shared many moments of excitement and disappointment and who encouraged me for the last ten years.

Finally, I would like to thank my family for their patience and support. I am grateful to my parents who have always stimulated my scientific development; to my sister for her constant reassurance and support and to my loving wife Diana, who gave me the greatest gift: our wonderful daughter Darina.

CONTENTS

INTRODUCTION

A vast research area in computer science is dedicated to establishing various properties of programs. While there are a lot of practical techniques in the realm of software engineering to test that a given program behaves as expected and uncover possible flaws, formal approaches to program correctness give mathematical guarantees that intensional features of a program translate to extensional characteristics. Some properties, such as type correctness or finite-state system soundness, are decidable and can be automatically affirmed for a given program. However, in many cases there is a need to establish the validity of an undecidable property, and in order to achieve this non-trivial evidence about the behaviour of the program needs to be presented. Such additional information can vary from annotations and hints on program components aiding logical inference to a complete and thorough mathematical proof. One of the most important features of a formal proof is that its validity can be verified objectively, systematically independently and efficiently. Thus, if we provide enough reasoning that a property holds for a given program, so that the soundness of the provided arguments is decidable, then the validity of the property can be automatically checked. The representation of the formal proof then acts as an unforgeable certificate that the property holds for the presented program. This concept is known as *proof-carrying code* [NL96].

The conventional method for obtaining certified code can be summarised as follows:

1. write a program P expressed in a formal language L,
2. write a specification A of a desired property in a logical system S extending L,
3. write a proof M of $A(P)$ in S.

Then the pair $\langle P, M \rangle$ is proof-carrying code. The first step is usually taken for granted, as an already written program P is observed. The second step can be more involved, but in practical situations usually a limited class of properties A is considered, usually referring to security or termination. The third step is most involved, as the proof M is not trivial to produce and, depending on the property A, can be exponentially larger than the proof M. An additional difficulty may be presented from the program P itself, if it is written in a style, which complicates the proof of its correctness. This problem can be mitigated by careful choice of the language L and by allowing changes to the program P in order to simplify the proof M. An instance of the latter approach is to use an annotated programming language L, which allows to insert minimal annotations in P, such that M can be automatically produced. A program, which is able to generate proof-carrying machine code from annotated source code is known as a *certifying compiler*.

The exciting discovery of the relation between constructive proofs and functional programs, which became known as the Curry-Howard isomorphism, has presented another possibility for producing proof-carrying code. A constructive proof C of a Π^0_2 statement $\forall x\, \exists y\, B(x, y)$ can be viewed as a computation of a function f, such that $\forall x\, B(x, f(x))$. In C there are two different kinds of computations interleaved: the computation of the value $f(x)$ and the computation of the validity of $B(x, f(x))$. Thus, when the proof C is instantiated with a value for x, it can be evaluated to obtain both a value $y = f(x)$ and a proof that $B(x, y)$ holds. Simple as it is, this approach is not optimal, as usually the correctness of the program is checked only once, instead of recomputing its validity for every supplied input x. This can be improved by applying Kreisel's *modified realisability* interpretation which splits the proof C into two distinct parts: the computational component (a program P) and the logical component (a proof M of its correctness). Thus we are able to automatically and efficiently obtain proof-carrying code from C, for the rather comprehensive property $A := \forall x\, B(x, P(x))$. This approach is commonly known as *program extraction*.

The feasibility of obtaining proof-carrying code for practical use is a topic of ongoing research [mob]. Currently, producing a program certified for more comprehensive properties, such as its complete functional specification, is not considered scalable. However, the problem is still important, as it can be used to create a fully certified library of commonly used and relatively small algorithms. In this case, program extraction might represent a viable alternative to the conventional methods for producing certified code.

The most difficult step in program extraction is to provide a constructive proof of the functional specification of the desired program. One possibility for simplifying this process is to allow the use of non-constructive reasoning. As it is well-known, Π^0_2 statements are equiderivable classically and constructively. Consequently, we should be able to obtain a correct program from a classical proof, which might be simpler and shorter than its constructive counterpart. However, what is not clear is whether the obtained program would be as readable and as efficient. Terms extracted from constructive proofs are essentially computational projections, hence they are shorter than the proof and moreover, we are able to control computational complexity by restricting used induction principles [BNS00, OW05]. If we have a proof, which is essentially a classical formulation of a constructive argument, we might still have a chance to recover the original construction. However, when there is non-trivial use of classical logic, the computational meaning of the proof is obscured and it is not as straightforward to reason about behaviour and efficiency of the underlying algorithm.

There exist a number of methods for obtaining programs reflecting the computational sense of non-constructive proofs. Even though they are based on different ideas for interpreting indirect reasoning, they have a common feature: additional computational infrastructure for supporting proof by contradiction is introduced. The following table summarises some of the currently available methods and the machinery that each of them employs.

Griffin's realisability	[Gri90]	control operators
$\mathbf{PA}_{\mathcal{C}}$	[BB93]	
Krivine's realisability	[Kri04]	
$\lambda\mu$-calculus	[Par92]	μ operator
(refined) A-translation	[Fri78, Dra80, Mur91, BBS02]	continuation passing style
Gödel's interpretation	[Göd58]	counterexample collection

Methods for extraction from non-constructive proofs

Most of the methods are concerned with obtaining some witness, claimed to exist by a given proof. However, there is no extensive research on how do the programs extracted via each of these methods relate with each other and what is the overhead introduced by the specific computational tools for modelling classical reasoning. One of the few such analyses known to the author has been carried out in [Mak06], where programs extracted from data-predicative proofs [Lei01] have been shown to have polynomial-time complexity under call-by-value and call-by-name reduction strategies. In the present work we compare two methods for extraction from non-constructive proofs: the refined A-translation and Gödel's Dialectica interpretation and show how the latter can be improved.

Chapter 1 defines three systems of arithmetic with finite types, which will be used to express functional programs and prove properties about them in a natural deduction style. The smallest of these systems is Negative Arithmetic, which is a negative formulation of Peano Arithmetic that admits classical reasoning and is used as a common ground for the application of the two extraction methods. A variant of this system is the Minimal Arithmetic, in which falsity is deliberately not defined using arithmetic means but taken as an uninterpreted predicate symbol. Finally, extending the language by adding a strong existential quantifier yields Heyting Arithmetic, in which constructive properties can be stated and proved.

Chapter 2 presents formal definitions of several proof interpretations and proves their soundness. Kreisel's modified realisability interpretation is used as the basis for the refined A-translation method, which translates proofs in Minimal Arithmetic into Heyting Arithmetic in order to extract their computational meaning. Gödel's Dialectica interpretation is defined in a natural deduction setting and acts on proofs in Negative Arithmetic, which can be obtained from Minimal Arithmetic by instantiating the abstract falsity with the arithmetic one.

Chapter 3 applies the methods to three case studies for non-constructive proofs: Stolzenberg's binary tape example, integer root of a unbounded function and the Infinite Pigeonhole Principle. The resulting programs are compared and analysed in terms of behaviour and time complexity. The analysis shows that the Dialectica interpretation can introduce significant computational overhead in comparison with refined A-translation. Several possibilities for improvement of counterexample collection are identified and addressed in the following chapters.

Chapter 4 reformulates the Dialectica interpretation in a way, such that common extracted subterms can be evaluated only once via a **let**-expression. This interpretation variant produces programs, which are of near linear size compared to the proof. The extracted terms are also more efficient since reevaluation of common subexpressions is avoided.

Chapter 5 extends the arithmetical system with uniform annotations that allow removing computations, which are irrelevant for the final result of the program. It is also demonstrated that by allowing a fine level of control of the computational meaning of the proofs, we are able to express the modified realisability interpretation within the Dialectica interpretation.

Chapter 6 identifies a special case of the induction principle, which is used in the considered case studies. The computational meaning of such a scheme can be given an alternative definition: an early aborting recursion, implemented via boolean flags expressing the validity of the current counterexample candidate. It is shown that this extension of the interpretation improves the average time complexity of the program extracted from the Infinite Pigeonhole Principle, thus making it asymptotically as efficient as its counterpart obtained via refined A-translation.

CHAPTER

ONE

SYSTEMS OF ARITHMETIC

In the present chapter we will define the basic logical notions, which will be used throughout the text. Our goal is to describe a system capturing both constructive and classical logic, built on top of an arithmetic with higher finite types. The arithmetical objects of the system will be considered as functional programs, while the logical objects will be the proofs, reasoning about such programs. A program extraction method will be considered as a meta-transformation Θ living outside the system, which translates a valid (possibly non-constructive) proof M of a certain formula A into a functional term t^{Θ} coupled with a valid proof M^{Θ} of a formula $A^{\Theta}(t^{\Theta})$, certifying that t^{Θ} is a "witness" possessing the properties described in A. The transformation Θ is traditionally called an *interpretation*, its domain is the *interpreted subsystem* and its range is the *verifying subsystem*.

An important requirement for this approach is that both the extracted program t^{Θ} and the certificate for its correctness M^{Θ} inhabit the same system as the original proof M. Consequently, even though that Θ and its general correctness are described on a meta-level, its input and output can still be formally implemented and verified in the same base system. This guarantees a certain level of safety: if we do not trust a certain meta-implementation of Θ, we can always verify the validity of its output by checking the extracted program t^{Θ} against the correctness certificate M^{Θ} via the same means, which we used to establish the correctness of the input proof M.

1.1 General notions

In this section we will define some general notions, which will be used throughout the text.

1.1.1 Inductive definitions

We start by stating some general facts about definitions by induction.

Definition 1.1 (Monotone mappings)**.** A mapping F over sets is *monotone* if whenever $X \subseteq Y$, we have $F(X) \subseteq F(Y)$.

Definition 1.2 (Inductively defined sets)**.** Let F be a monotone mapping over sets. The set X is *inductively defined* by F if X is the least fixed point of F, i.e., the least set with respect to the inclusion relation $\subseteq$, such that $F(X) = X$. The unique existence of X is guaranteed by Knaster–Tarski's theorem.

Very often we will work in an extended setting, where we define a finite number of syntactic objects by induction simultaneously.

Definition 1.3 (Simultaneous induction)**.** Let $X = X_1 \times X_2 \times \ldots \times X_n$. If X is inductively defined by F, then we say that $X_1, X_2, \ldots X_n$ are defined by F through *simultaneous induction.*

For simultaneous inductive definitions we will usually define the monotone F extensionally using a conjunction of properties (*inductive clauses*) P of the form

> "if x_0 is in T and $\overrightarrow{x_1}$ are in X_1 and $\ldots$ and $\overrightarrow{x_n}$ are in X_n then $f(\vec{x})$ is a member of the i-th component of $F(\vec{X})$"

for some $i \in \{1, \ldots, n\}$, some set of external objects T and some mapping f of appropriate arity, where $\overrightarrow{x_j}$ are vectors of meta-variables. Intuitively, f is a rule specifying how to construct an object in X_i from a number if previously constructed objects in $\vec{X}$. Also, for convenience we can think that all argument positions over which f is constant are omitted from its signature. It can be seen that the above form of inductive clauses guarantees the monotonicity of F. Indeed, any element of $F(\overrightarrow{X})$ must be constructed via a clause P, whose premises are monotone with respect to all X_j.

We will exclusively work with non-empty countable sets of finitely representable syntactic objects. Thus we can fix the set of natural numbers as the universe in which these syntactic objects are encoded. Unless stated otherwise, we assume that newly defined objects do not have the same representation as any of the previously defined ones. We will mostly use inductive definitions to define syntactic objects and transformations over them. It is thus convenient to view the functions f appearing in the clauses P as syntactic constructors and will require them to satisfy the following "constructor" conditions:

- $f(\overrightarrow{x}) > \max(\overrightarrow{x})$ for all $\overrightarrow{x}$

- $\overrightarrow{x_1} \neq \overrightarrow{x_2}$ implies $f(\overrightarrow{x_1}) \neq f(\overrightarrow{x_2})$ for all $\overrightarrow{x_i}$, i.e., f is injective;
- all considered f have disjoint ranges.

The conditions above guarantee that every inductive step generates intensional objects with fresh and unique representations. In particular, for every syntactic object there can be identified a unique inductive clause and a corresponding constructor f, which generated it.

Next, we will define transformations on syntactic objects by "induction on the definition" of their domain.

Definition 1.4 (Induction on the definition)**.** Let $X_1, \ldots, X_n$ be mutually disjoint sets defined inductively using a set of clauses of the form P. Let $O = \bigcup_{i=1}^{n} X_i$, let Y be an arbitrary parameter set, an let Z be an arbitrary result set. We say that the mapping $\phi : O \times Y \to Z$ is defined *by induction on the definition of* $\overrightarrow{X}$ if there are mappings $\phi_i : X_i \times Y \to Z$ for $i \in \{1, \ldots, n\}$ such that $\phi = \bigcup_{i=1}^{n} \phi_i$ and whose graphs G_i are defined through simultaneous induction by a set of clauses obtained by replacing every clause P by the clause $P_{\Phi,\Psi}$ as follows:

> "if x_0 is in T and $\overrightarrow{(x_1, \Psi(x_0, y), z_1)}$ are in G_1 and ... and $\overrightarrow{(x_n, \Psi(x_0, y), z_n)}$ are in G_n then $(f(\vec{x}), y, \Phi(x_0, y, \vec{z}))$ is a member of the i-th component of $F(\vec{G})$",

where Φ, Ψ are clause-specific mappings and y is a meta-variable ranging over Y.

Intuitively, Φ is a rule, specifying how to construct the result of the function ϕ_i when applied to a complex object $f(\vec{x})$ by using the values of ϕ_i on the components $\vec{x}$, which were used to construct the object. The parameter set Y allows us to define binary functions by induction on one of the arguments, and the mapping Ψ is used to specify for which instances of the other argument do we use the induction hypothesis. Unless defined otherwise, by default we assume that Y is trivial (a singleton set) and that for each clause $\Phi := f$. Note that if nothing else is specified, the resulting mappings ϕ_i are exactly the identity mappings on X_i.

The constructor conditions for f guarantee that the mappings ϕ_i are correctly defined and with disjoint domains, which implies that ϕ is a correctly defined mapping.

1.1.2 Variables and substitutions

Variables and substitutions will be ground concepts in our syntactic definitions and in this section we will present some general definitions to unify their treatment. Below we will assume that we have a fixed set of inductive clauses, defining simultaneously some family of (disjoint) sets of objects $X_1, \ldots, X_n$. We denote the set of all objects $O := \bigcup_{i=1}^{n} X_i$. All mappings will be defined by induction on the definition of $\vec{X}$.

We will adopt the convention that there will be a preliminarily fixed countable set of variables V, which will be used in the generation of the set of syntactic objects O.

Definition 1.5 (Variable clause)**.** An inductive clause is called a *variable clause* if it constructs an object from a variable. Formally, we require that the parameter set is exactly the set of variables, i.e., $T = V$, and f depends only on the parameter x_0.

Definition 1.6 (Variable-binding clause)**.** An inductive clause is called a *variable-binding clause* if it includes a variable in the construction of the object. Formally, we require that $T = V$ and f **does not depend only** on x_0.

Definition 1.7 (Free variables)**.** The mapping $\mathsf{FV} : O \to \mathcal{P}(V)$ is defined by taking

- $\Phi_1(x_0, \vec{z}) := \{x_0\}$ for every variable clause,
- $\Phi_2(x_0, \vec{z}) := \bigcup \vec{z} \setminus \{x_0\}$ for every variable-binding clause,
- $\Phi_3(x_0, \vec{z}) := \bigcup \vec{z}$ for every other clause.

$\mathsf{FV}(x)$ is the set of *free variables* of x.

Definition 1.8 (Bound variables)**.** The mapping $\mathsf{BV} : O \to \mathcal{P}(V)$ is defined by taking

- $\Phi_1(x_0, \vec{z}) := \bigcup \vec{z} \cup \{x_0\}$ for every variable-binding clause,
- $\Phi_2(x_0, \vec{z}) := \bigcup \vec{z}$ for every other clause.

$\mathsf{BV}(x)$ is the set of *bound variables* of x.

Definition 1.9 (Closed objects)**.** We say that an object x is *closed* if $\mathsf{FV}(x) = \emptyset$.

We will extend the definition of free and bound variables to sets of object using the following notation:

$$\mathsf{FV}[X] := \bigcup_{x \in X} \mathsf{FV}(x), \qquad \mathsf{BV}[X] := \bigcup_{x \in X} \mathsf{BV}(x).$$

Definition 1.10 (Substitution)**.** A *substitution* is a partial mapping $\sigma : V \dashrightarrow O$. In case $\mathrm{dom}(\sigma)$ is finite, we will denote σ by $[x_1, \ldots, x_n := \sigma(x_1), \ldots, \sigma(x_n)]$, where $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$.

Definition 1.11 (Substitution application)**.** Let σ be a substitution. We define *restricted substitution application* as a binary mapping $\Sigma : O \to \mathcal{P}(V) \to O$, where the second argument is meant to be a set of bound variables that the substitution should ignore. Σ is defined by induction on the definition of O with the mapping

$\Psi(x_0, y) := y \cup \{x_0\}$ for every variable-binding clause, which is used to update the set of bound variables, and the mapping

$$\Phi(x_0, y, \vec{z}) := \begin{cases} \sigma(x_0), & \text{if } x_0 \in \mathrm{dom}(\sigma) \setminus y, \\ f(x_0, \vec{z}), & \text{otherwise,} \end{cases}$$

for every variable clause, which actually applies the substitution. The *(full) substitution application* of σ to x will be denoted as $x\sigma$, and is defined as $x\sigma := \Sigma(x, \emptyset)$.

Definition 1.12 (Capture-free substitution)**.** We call the substitution application $x\sigma$ *capture-free* if none of the objects substituted for free variables in x by σ have a free variable, which is bound in x. Formally, we require that $\mathsf{BV}(x) \cap \mathsf{FV}(\sigma(y)) = \emptyset$ for all $y \in \mathsf{FV}(x) \cap \mathrm{dom}(\sigma)$.

Definition 1.13 (Variable renaming)**.** We call the substitution υ *a variable renaming* if $\mathrm{ran}(\upsilon) \subseteq V$.

Definition 1.14 (Renaming of bound/free variables)**.** Let υ be a variable renaming. We define a variation of the substitution application, which we call *renaming of bound variables.* The mapping $\Upsilon : O \to \mathcal{P}(V) \to O$ is defined by induction on the definition of O with the mapping $\Psi(x_0, y) := y \cup \{x_0\}$ for every variable-binding clause and the mapping

$$\Phi(x_0, y, \vec{z}) := \begin{cases} f(\sigma(x_0), \vec{z}), & \text{if } x_0 \in \mathrm{dom}(\sigma) \cap y, \\ f(x_0, \vec{z}), & \text{otherwise,} \end{cases}$$

for every variable and variable-binding clause. We will use $x\sharp\upsilon$ to denote $\Upsilon(x, \emptyset)$. For any υ we call $x\sharp\upsilon$ a *variant* of x. It is easy to see that "is a variant of" is an equivalence relation (traditionally also called α-equivalence).

If we change the condition in Φ to "if $x_0 \in \mathrm{dom}(\sigma) \setminus y$", then we obtain a mapping, which we call *renaming of free variables.*

It is well-known that capture-free substitution applications exhibit nice preservation properties, which are not generally true if we allow variable capturing.

Proposition 1.15. *Let σ be a substitution, x be an object and let $x\sigma$ be capture-free.*

1. $\mathsf{FV}(x\sigma) = \bigcup_{y \in \mathsf{FV}(x) \cap \mathrm{dom}(\sigma)} \mathsf{FV}(\sigma(y)) \cup \big(\mathsf{FV}(x) \setminus \mathrm{dom}(\sigma)\big)$,
2. $\mathsf{BV}(x\sigma) = \bigcup_{y \in \mathsf{FV}(x) \cap \mathrm{dom}(\sigma)} \mathsf{BV}(\sigma(y)) \cup \mathsf{BV}(x)$,
3. *If x' is a variant of x, such that $x'\sigma$ is capture-free, then $x'\sigma$ is a variant of $x\sigma$.*

It is clear that for every syntactic object x and substitution σ we can find a variant x' of x, such that $x'\sigma$ is capture-free. Moreover, object variants are indiscernible with respect to capture-free substitution applications. Thus, unless stated otherwise, we will assume that we always silently take appropriate variants of objects, such that substitution applications are capture-free.

1.2 Arithmetical notions

We will base our logical systems on an arithmetic of higher finite types, with primitive recursion, which is also the basis for Gödel's well-known system T [Göd58]. We start by defining base and finite types by simultaneous induction.

Definition 1.16 (Finite types)**.** Base types are

- α, denoting a *finite type variable* from a fixed countable list of type variables;
- B, denoting the type of *booleans*;
- N, denoting the type of *natural numbers*;
- $\mathsf{L}(\rho)$, denoting a type of *lists*, where ρ is the base type of the list elements.

Finite types are

- μ, where μ is a base type;
- $\rho \Rightarrow \sigma$, denoting an *arrow type*, where ρ and σ are finite types;
- $\rho \times \sigma$, denoting a *product type*, where ρ and σ are finite types.

Note that we allow lists to have elements only of base type. In what follows only finite types will be considered, so the word "finite" will be omitted. The complexity of finite types is measured by their degree.

Definition 1.17 (Type degree)**.** Let τ be a type without free type variables. We define its *degree* by induction as follows:

- $\deg(\mu) := 0$ for μ a base type,
- $\deg(\rho \Rightarrow \sigma) := \max(\deg(\rho) + 1, \deg(\sigma))$,
- $\deg(\rho \times \sigma) := \max(\deg(\rho), \deg(\sigma))$.

Next, we will define the terms in our system. Every term t will have a unique type τ explicitly associated with it. This is commonly known as "Church style" typing. In the setting of Section 1.1.2 term variables and constants will have the set of types as external parameters and terms will be defined simultaneously with their typing. We will use the typing notations t^ρ and $t : \rho$.

Definition 1.18 (Terms)**.** Terms are

- x^ρ, denoting a *typed object variable* chosen from a fixed list of object variables;
- $(s^{\rho\Rightarrow\sigma} t^\rho)^\sigma$, denoting an *application* of the function s to an argument t;
- $(\lambda x^\rho\, t^\sigma)^{\rho\Rightarrow\sigma}$, denoting a *binding abstraction* of the variable x from the term t;
- $\mathsf{Pair}_{\rho,\sigma}^{\rho\Rightarrow\sigma\Rightarrow\rho\times\sigma}$, denoting a *pair constructor*;
- tt^B, denoting the boolean constant for *truth*;
- ff^B, denoting the boolean constant for *falsity*;

- 0^{N}, denoting the natural constant for *zero*;
- $\mathsf{S}^{\mathsf{N}\Rightarrow\mathsf{N}}$, denoting the constant function for *successor*;
- $(\mathsf{nil}_\rho)^{\mathsf{L}(\rho)}$, denoting the *empty list* constant for lists of type ρ;[1]
- $(::_\rho)^{\rho\Rightarrow\mathsf{L}(\rho)\Rightarrow\mathsf{L}(\rho)}$, denoting the *constructor* function for lists of type ρ[1];
- $\mathsf{Split}^\tau_{\rho,\sigma} : \rho\times\sigma \Rightarrow (\rho\Rightarrow\sigma\Rightarrow\tau)\Rightarrow\tau$, denoting the *pair splitting* for type τ;
- $\mathsf{Cases}^\tau : \mathsf{B}\Rightarrow\tau\Rightarrow\tau\Rightarrow\tau$, denoting the *if-then-else* constant for type τ;
- $\mathcal{R}^\tau_{\mathsf{N}} : \mathsf{N}\Rightarrow\tau\Rightarrow(\mathsf{N}\Rightarrow\tau\Rightarrow\tau)\Rightarrow\tau$, denoting the *recursor for natural numbers* for type τ;
- $\mathcal{R}^\tau_{\mathsf{L}(\rho)} : \mathsf{L}(\rho)\Rightarrow\tau\Rightarrow(\rho\Rightarrow\mathsf{L}(\rho)\Rightarrow\tau\Rightarrow\tau)\Rightarrow\tau$, denoting the *recursor for lists of type* ρ for type τ.

We will use the following shortcut notations for the pairing and projection operations:

$$\begin{aligned} \langle s,t\rangle &:= \mathsf{Pair}\,s\,t, & t_{\llcorner} &:= \mathsf{Split}\,t\,(\lambda x\,\lambda y\,x),\\ \langle r,s,t\rangle &:= \langle r,\langle s,t\rangle\rangle\,, & t_{\lrcorner} &:= \mathsf{Split}\,t\,(\lambda x\,\lambda y\,y). \end{aligned}$$

The notions of free and bound variables and substitutions from Section 1.1.2 are naturally applicable to terms and types. In addition, note that a type substitution naturally extends to terms. Indeed, if we consider the type components of the terms as “variables” and the type substitution as a “variable renaming”, then the type substitution applied to a term t would be “renaming of the free variables” of t.

All finite types are inhabited, i.e., there is a closed term t^ρ for each type ρ.

Definition 1.19 (Canonical inhabitant)**.** For each closed finite type ρ we define a closed term of type ρ, called the *canonical inhabitant* of ρ and denoted as $\Box^\rho$.

$$\Box^{\mathsf{B}} := \mathsf{ff},\qquad \Box^{\mathsf{N}} := 0,\qquad \Box^{\mathsf{L}(\rho)} := \mathsf{nil}_\rho,$$
$$\Box^{\rho\Rightarrow\sigma} := \lambda x^\rho\Box^\sigma,\qquad \Box^{\rho\times\sigma} := \langle\Box^\rho,\Box^\sigma\rangle\,.$$

1.2.1 Term reduction and normalization

The operational semantics of the term system are inductively defined via the following reduction rules.

Definition 1.20 (Term reductions)**.**

$$\begin{aligned} &(\lambda x\,s)t &&\mapsto\; s\,[x:=t]\,, && \mathcal{R}_{\mathsf{N}}\,0\,s\,t &&\mapsto\; s,\\ &\mathsf{Split}\,\langle s,t\rangle\,f &&\mapsto\; f\,s\,t, && \mathcal{R}_{\mathsf{N}}\,(\mathsf{S}n)\,s\,t &&\mapsto\; t\,n\,(\mathcal{R}_{\mathsf{N}}\,n\,s\,t),\\ &\mathsf{Cases}\,\mathsf{tt}\,s\,t &&\mapsto\; s, && \mathcal{R}_{\mathsf{L}(\rho)}\,\mathsf{nil}\,s\,t &&\mapsto\; s,\\ &\mathsf{Cases}\,\mathsf{ff}\,s\,t &&\mapsto\; t, && \mathcal{R}_{\mathsf{L}(\rho)}\,(n::l)\,s\,t &&\mapsto\; t\,n\,l\,(\mathcal{R}_{\mathsf{L}(\rho)}\,l\,s\,t),\\ &(\lambda x\,sx) &&\mapsto\; s, && \text{if } s \text{ not an abstraction and } x\notin\mathsf{FV}(s), \end{aligned}$$

[1]For readability we will usually skip the subscript ρ for the constant nil and the constructor $(::)$.

and if $s \mapsto s'$, then

$$sr \mapsto s'r, \qquad rs \mapsto rs', \qquad \lambda x\, s \mapsto \lambda x\, s'.$$

From the definition it is immediate that the reduction $\mapsto$ is type-preserving. Sometimes for convenience we will use a "let" notation for a β-redex and "pair" abstraction:

$$\begin{aligned} \textbf{let } x := t \textbf{ in } s &\quad := \quad (\lambda x\, s)t, \\ \lambda\langle x, y\rangle\, t &\quad := \quad \lambda z \left(\textbf{let } x := z_{\llcorner} \textbf{ in } \textbf{let } y := z_{\lrcorner} \textbf{ in } t\right). \end{aligned}$$

Definition 1.21 (Multiple-step reduction)**.** We say that a term s reduces in multiple steps to t, denoted as $s \overset{*}{\mapsto} t$, if

1. $s \equiv t$, or
2. $s \mapsto r \overset{*}{\mapsto} t$ for some term r.

Definition 1.22 (Set of reducts)**.** The *set of reducts* of a term s is defined as

$$\mathrm{Red}_s := \left\{t :\ s \overset{*}{\mapsto} t\right\}.$$

Definition 1.23 (Normal form)**.** We say that a term s is in *normal form* if it is irreducible, i.e., $\mathrm{Red}_s = \{s\}$.

Definition 1.24 (Strongly normalizing)**.** We say that a term s is strongly normalizing if Red_s is finite.

The proofs of strong normalization and confluence of $\mapsto$ can be found in [SW10].

Theorem 1.25 (Strong normalization)**.** *Every term is strongly normalizing.*

Theorem 1.26 (Confluence)**.** *If $r \overset{*}{\mapsto} s_1$ and $r \overset{*}{\mapsto} s_2$, then there is a term t, such that $s_1 \overset{*}{\mapsto} t$ and $s_2 \overset{*}{\mapsto} t$.*

Corollary 1.27. *Every term has a unique normal form.*

Remark 1.28. The reduction relation $\mapsto$ considered here is by far not an optimal one. For example, it does not capture any simplifications or permutative conversions like:

$$\begin{aligned} &\langle r_{\llcorner}, r_{\lrcorner}\rangle && \mapsto r, \\ &\mathsf{Cases}\, b\, \mathsf{tt}\, \mathsf{ff} && \mapsto b, \\ &(\mathsf{Cases}\, b\, r\, s)t && \mapsto \mathsf{Cases}\, b\, (rt)\, (st), \\ &\langle \mathsf{Cases}\, b\, r_1\, s_2, \mathsf{Cases}\, b\, r_2\, s_2\rangle && \mapsto \mathsf{Cases}\, b\, \langle r_1, r_2\rangle\, \langle s_1, s_2\rangle\,. \end{aligned}$$

Additionally, in some cases it is useful to consider η-expansions instead of η-reductions by regarding the η-long normal form of terms (cf. [BES98, BES03]). In general, for practical considerations the reduction relation can be weakened so that we obtain less and larger term equivalence classes. For example, the above mentioned improvements to the reduction relation are implemented in the interactive proof assistant MINLOG. However, for our theoretical needs the stronger reduction relation from Definition 1.20 will be sufficient.

1.2.2 Removing empty computational content

During the presentation we will encounter cases in which we would like to denote lack of computational information. For this we will use a special singleton (unit) base type, denoted as I, which will contain a single constant, denoted as ε^{I}. Since our term system has no side effects, we will not be interested in computations involving terms of I. We will define reductions, which remove all possible uses of the type I and terms of this type.

Definition 1.29 (I-reduction)**.**

$$\begin{array}{lcl@{\qquad\qquad}lcl} \rho \times \mathsf{I} & \overset{\mathsf{I}}{\mapsto} & \rho, & \rho \Rightarrow \mathsf{I} & \overset{\mathsf{I}}{\mapsto} & \mathsf{I}, \\ \mathsf{I} \times \rho & \overset{\mathsf{I}}{\mapsto} & \rho, & \mathsf{I} \Rightarrow \rho & \overset{\mathsf{I}}{\mapsto} & \rho, \end{array}$$

For presentational convenience we assume that these type reductions are always implicitly executed when forming a product type or an arrow type involving I. As a result, we will never be able to construct a complex type containing I. A side effect from this convention is that the operations $\times$ and $\Rightarrow$ do not behave as syntactic constructors any more, because they can produce types smaller than their operands. For this reason, when we are given a type ρ, we will not be able to determine if it is a result of a construction involving I or not. Consequently, when we argue by induction on the definition of ρ we will inevitably skip all construction steps involving I. However, in our setting this will pose no problems, as these skipped constructions will be in fact computationally irrelevant.

The implicit removal of the type I causes some awkward effects on term construction. For example, $s^{\mathsf{I}}t^{\rho}$, $t^{\rho}s^{\mathsf{I}}$, $t^{\rho}\llcorner$ are now valid term constructions for any type ρ. We avoid such anomalies by defining another reduction acting on the term level.

Definition 1.30 (ε-reduction)**.**

$$s^{\mathsf{I}} \overset{\varepsilon}{\mapsto} \varepsilon,$$

for s a variable or one of $\mathsf{Pair}_{\mathsf{I},\mathsf{I}}, \mathsf{Split}^{\mathsf{I}}_{\rho,\sigma}, \mathsf{Cases}^{\mathsf{I}}, \mathcal{R}^{\mathsf{I}}_{\mathsf{N}}, \mathcal{R}^{\mathsf{I}}_{\mathsf{L}(\rho)}$,

$$\begin{array}{lll@{\qquad\qquad}lll}
s^\rho \varepsilon & \overset{\varepsilon}{\mapsto} & s^\rho, & \lambda x^{\mathsf{I}}\, s^\rho & \overset{\varepsilon}{\mapsto} & s^\rho, \\
\varepsilon s^\rho & \overset{\varepsilon}{\mapsto} & \varepsilon, & \lambda x^\rho\, \varepsilon & \overset{\varepsilon}{\mapsto} & \varepsilon, \\
\mathsf{Pair}_{\rho,\mathsf{I}} & \overset{\varepsilon}{\mapsto} & \lambda x^\rho\, x, & \mathsf{Split}^{\tau}_{\rho,\mathsf{I}} & \overset{\varepsilon}{\mapsto} & \lambda x^\rho\, \lambda f^{\rho\Rightarrow\tau}\, fx, \\
\mathsf{Pair}_{\mathsf{I},\rho} & \overset{\varepsilon}{\mapsto} & \lambda x^\rho\, x, & \mathsf{Split}^{\tau}_{\mathsf{I},\rho} & \overset{\varepsilon}{\mapsto} & \lambda x^\rho\, \lambda f^{\rho\Rightarrow\tau}\, fx.
\end{array}$$

As before, we assume that the reduction $\overset{\varepsilon}{\mapsto}$ is implicitly executed when constructing a term involving I. It is easy to see that we cannot construct a complex term involving ε. Moreover, ε is the only term of type I that can be constructed. With $\overset{\varepsilon}{\mapsto}$ we restore the parity between terms and their types, so that whenever a complex type construction collapses because it involves I, on the term level we also ignore the corresponding construction step involving ε.

Remark 1.31. For completeness, we could also postulate

$$\begin{array}{lll@{\qquad\qquad}lll}
\mathsf{L}(\mathsf{I}) & \overset{\mathsf{I}}{\mapsto} & \mathsf{N}, & \mathsf{nil}_{\mathsf{I}} & \overset{\varepsilon}{\mapsto} & 0, \\
\mathcal{R}^{\tau}_{\mathsf{L}(\mathsf{I})} & \overset{\varepsilon}{\mapsto} & \mathcal{R}^{\tau}_{\mathsf{N}}, & ::_{\mathsf{I}} & \overset{\varepsilon}{\mapsto} & \mathsf{S}.
\end{array}$$

However, for the sake of clarity, we will never make use of the list type with I as a parameter.

Remark 1.32. The currently presented approach for removing computationally irrelevant terms cleans eagerly on each construction step. This technique is employed also in [Sch08, SW10]. An alternative approach for removing computationally irrelevant terms and types is to use I and ε explicitly during extraction and to have an additional cleaning step, which performs all possible reductions simultaneously on the extracted term and its type. An example of this method is the erase function of [Mak06]. The author believes that the approach chosen here possesses some advantages over its alternative: it is much cleaner notationally and is more memory efficient when implemented.

1.3 Logical systems

In this section we will will start the definition of three systems HA^ω (Heyting Arithmetic with finite types) and its negative fragments MA^ω (Minimal Arithmetic) and NA^ω (Negative Arithmetic). The system HA^ω is essentially the same as in [Tro73]. The systems MA^ω and NA^ω will obey to the same logical rules as HA^ω, but their language will be restricted to negative formulas only. The difference between the two systems will be in the definition of falsity and negation, which will result in systems of different strength.

1.3.1 Formulas

We start by defining the formula language of the three systems NA^ω, MA^ω and HA^ω.

Definition 1.33 (Formulas)**.** *Formulas* of NA^ω are

- $\mathrm{at}(t^{\mathsf{B}})$, denoting a *decidable atomic formula*, where t^{B} is an arbitrary boolean term;
- $A \to B$, denoting an *implication* from A to B;
- $\forall x^\rho\, A$, denoting *universal quantification* of A over the variable x with $\rho \neq \mathsf{I}$.

The formulas of MA^ω are obtained by adding to NA^ω the clause

- $\bot$, a *predicate variable* used for denoting *falsity*.

The formulas of HA^ω are obtained by adding to NA^ω the clauses

- $A \wedge B$, denoting a *conjunction* of A and B;
- $\exists x^\rho\, A$, denoting *existential quantification* of A over the variable x with $\rho \neq \mathsf{I}$.

We will denote the quantifier-free fragments of the systems NA^ω, MA^ω and HA^ω as NA^ω_0, MA^ω_0 and HA^ω_0 respectively. The atomic fragments of NA^ω and HA^ω coincide and we will denote the resulting system as $\mathrm{NA}^\omega_{\mathrm{at}}$.

The notions of free and bound variables and substitutions from Section 1.1.2 are transferable to formulas. In addition, term and type substitutions can also be extended to formulas if we consider the term parameter in the atomic formula as "a variable" and the corresponding substitutions as "variable renamings".

In formulas of MA^ω we have an additional kind of variable: the predicate variable $\bot$. We reserve the notations FV and BV for free and bound term variables. We will call a MA^ω formula $\bot$-*free* if it does not contain $\bot$. In fact, $\bot$-free formulas of MA^ω are exactly the formulas of NA^ω. We also have a corresponding notion of *formula substitution*, which substitutes a formula for the predicate variable $\bot$. As usual, we assume that all considered formula substitutions are capture-free with respect to variables bound by universal quantifiers.

For technical convenience, we extend the definition of universal quantification to I as follows:

$$\forall x^{\mathsf{I}} A \overset{\mathsf{I}}{\mapsto} A.$$

The definition is sound, as due to the $\overset{\varepsilon}{\mapsto}$ reduction, A cannot contain the variable x freely.

The $\mathrm{at}(\cdot)$ construction is intended to embed boolean terms into the logical world. In this way the logical systems contain all predicates expressible in Gödel's system T.

These are exactly the predicates, which can be proved total by transfinite induction up to ε_0 [Kre51, Kre52, AF98].

The following notion of subformula is due to Gentzen.

Definition 1.34 (Subformula)**.** For a formula C we define the notions of a negative, positive and strictly positive *subformula*.

- C is a positive and strictly positive subformula of itself;
- C is a negative/positive/strictly positive formula of $A \wedge B$ if C is a negative/positive/strictly positive subformula of A or B;
- C is a negative/positive formula of $A \to B$ if C is a negative/positive subformula of A or a positive/negative subformula of B. C is a strictly positive subformula of $A \to B$, if C is a strictly positive subformula of B;
- C is a negative/positive/strictly positive formula of $\forall x^\rho\, A$ or $\exists x^\rho\, A$ if C is a negative/positive/strictly positive of $A\,[x := t]$ for some term t^ρ.

Falsity in minimal logic systems is traditionally a symbol without any logical meaning attached to it [Joh37, Pra71, TS00]. In MA^ω this ultimate generality is expressed by defining $\bot$ to be a predicate variable, a placeholder for an arbitrary formula. We can use $\bot$ to define negation in MA^ω and then employ it to define negative versions of the connectives of HA^ω.

Definition 1.35 (Negative connectives in MA^ω)**.** The *negative connectives in* MA^ω are

- $\neg A := A \to \bot$, denoting *negation* of A;
- $A \tilde{\wedge} B := \neg(A \to B \to \bot)$, denoting *weak conjunction* of A and B;
- $\tilde{\exists} x^\rho\, A := \neg(\forall x \neg A)$, denoting *weak existential quantification* of A on x.

The systems HA^ω and NA^ω are meant to describe intuitionistic and classical arithmetic with finite types. In these systems we would like a notion of falsity F, which validates the schemata:

$$\begin{array}{lcl} \textit{ex falso quodlibet}: & \mathrm{F} \to A & \text{for any } \mathrm{HA}^\omega \text{ formula } A, \\ \textit{stability}: & ((A \to \mathrm{F}) \to \mathrm{F}) \to A & \text{for any } \mathrm{NA}^\omega \text{ formula } A. \end{array}$$

Since our system is entirely based on decidable atomic formulas, it seems natural to define falsity using the underlying arithmetic. We can then translate the negative connectives of MA^ω to NA^ω.

Definition 1.36 (Arithmetical falsity and truth)**.** The *arithmetical falsity* in NA^ω is defined as $\mathrm{F} := \mathrm{at}(\mathsf{ff})$ and the *arithmetical truth* in NA^ω is defined as $\mathrm{T} := \mathrm{at}(\mathsf{tt})$. The *negative connectives in* NA^ω are obtained from the corresponding negative connectives of MA^ω by substituting F for $\bot$.

In the next section we will show that the arithmetical falsity indeed satisfies the above axiom schemata in NA^ω and HA^ω.

Using $\mathrm{at}(\cdot)$, we can define equality at the base types by defining appropriate boolean programs. We extend this equality extensionally to higher types.

Definition 1.37 (Arithmetical equality)**.** For every closed type τ we define an *equality predicate* inductively as follows:

$$
\begin{aligned}
(x \overset{\mu}{=} y) &:= \mathrm{at}(\mathrm{Eq}_\mu xy), \text{ for } \mu \text{ a base type, where}\\
\mathrm{Eq}_{\mathsf{B}} &:= \lambda x^{\mathsf{B}}\, \lambda y^{\mathsf{B}}\, \mathsf{Cases}\, x\, y\, (\mathsf{Cases}\, y\, \mathsf{ff}\, \mathsf{tt}),\\
\mathrm{Eq}_{\mathsf{N}} &:= \lambda x^{\mathsf{N}}\, \mathcal{R}_{\mathsf{N}}^{\mathsf{N}\Rightarrow\mathsf{B}} x \big(\lambda y^{\mathsf{N}}\, \mathcal{R}_{\mathsf{N}}^{\mathsf{B}}\, y\, \mathsf{tt}(\lambda y_0^{\mathsf{N}}\, \lambda q^{\mathsf{B}}\, \mathsf{ff})\big)\\
&\qquad\qquad \big(\lambda x_0^{\mathsf{N}}\, \lambda p^{\mathsf{N}\Rightarrow\mathsf{B}}\, \lambda y^{\mathsf{N}}\, \mathcal{R}_{\mathsf{N}}^{\mathsf{B}}\, y\, \mathsf{ff}(\lambda y_0^{\mathsf{N}}\, \lambda q^{\mathsf{B}}\, p y_0)\big),\\
\mathrm{Eq}_{\mathsf{L}(\rho)} &:= \lambda x^{\mathsf{L}(\rho)}\, \mathcal{R}_{\mathsf{L}(\rho)}^{\mathsf{L}(\rho)\Rightarrow\mathsf{B}}\, x\, \big(\lambda y^{\mathsf{L}(\rho)}\, \mathcal{R}_{\mathsf{L}(\rho)}^{\mathsf{B}}\, y\, \mathsf{tt}\, (\lambda y_h^\rho\, \lambda y_t^{\mathsf{L}(\rho)}\, \lambda q^{\mathsf{B}}\, \mathsf{ff})\big)\\
&\quad \big(\lambda x_h^\rho\, \lambda x_t^{\mathsf{L}(\rho)}\, \lambda p^{\mathsf{L}(\rho)\Rightarrow\mathsf{B}}\, \lambda y^{\mathsf{L}(\rho)}\, \mathcal{R}_{\mathsf{L}(\rho)}^{\mathsf{B}}\, y\, \mathsf{ff}(\lambda y_h^\rho\, \lambda y_t^{\mathsf{L}(\rho)}\, \lambda q^{\mathsf{B}}\, \mathsf{Cases}(\mathrm{Eq}_\rho x_h y_h)(p y_t)\mathsf{ff})\big),\\
(x \overset{\rho\times\sigma}{=} y) &:= (x_{\llcorner} \overset{\rho}{=} y_{\llcorner})\, \tilde{\wedge} (x_{\lrcorner} \overset{\sigma}{=} y_{\lrcorner}),\\
(x \overset{\rho\Rightarrow\sigma}{=} y) &:= \forall z^\rho\, (xz \overset{\sigma}{=} yz).
\end{aligned}
$$

We need to express a notion of *evaluational equality*, or equality up to normal form. However, we prefer to keep term evaluation as an arithmetical concept and not transfer it to the logical world. In order to avoid adding reduction axioms to the logical system, we extend the notion of syntactic equality of formulas, by postulating that two atomic formulas are considered the same if their terms reduce to the same normal form. As already noted in Section 1.2.1, terms are strongly normalizing, so the definition is legal. Thus, throughout the text three different notions of term equaility will be used:

- $t_1 \equiv t_2$, denoting *syntactical equality*, i.e., that t_1 and t_2 are syntactically the same,
- $t_1 \overset{r}{=} t_2$, denoting *evaluational equality*, i.e., that t_1 and t_2 have the same normal form,
- $t_1 = t_2$, denoting *logical equality*, i.e., that t_1 and t_2 are provably equal in a given system of arithmetic.

1.3.2 Proofs

We will conclude the exposition of the systems MA^ω, NA^ω and HA^ω by defining the valid derivations in them. We will work in a natural deduction setting and the derivations will use a λ-syntax similar to that of terms in order to stress the Curry-Howard correspondence. These proof terms will be built from a countable set of *assumption variables* and will be typed by the formulas they prove. We will define proof terms and their "typing" by simultaneous induction.

Definition 1.38 (Proof terms)**.** The proof terms of NA^ω are

- u^A, denoting an *assumption* u of a formula A;
- $\mathsf{AxT} : \mathrm{T}$, denoting the *truth axiom*, which gives the semantics of $\mathrm{at}(\cdot)$;
- $(\lambda u^A\, M^B)^{A \to B}$, denoting the *implication introduction* rule discharging the assumption u from the proof M;
- $(M^{A \to B} N^A)^B$, denoting the *implication elimination* binary rule;
- $(\lambda x^\rho\, M^A)^{\forall x^\rho A}$, denoting the *universal introduction* rule binding the variable x. This rule is subjected to the usual variable condition that $x \notin \mathsf{FV}[\mathsf{FA}(M)]$;
- $(M^{\forall x^\rho A} t^\rho)^{A[x:=t]}$, denoting the *universal elimination* rule;
- $\mathcal{C}^{b,A} : \forall b^{\mathsf{B}}\, (A\,[b := \mathsf{tt}] \to A\,[b := \mathsf{ff}] \to A)$, denoting the *boolean case distinction* axiom for the formula A;
- $\mathsf{Ind}_{\mathsf{N}}^{n,A} : \forall n^{\mathsf{N}}\, (A\,[n := 0] \to \forall n^{\mathsf{N}}\, (A \to A\,[n := \mathsf{S}n]) \to A)$, denoting the *natural number induction* axiom for the formula A;
- $\mathsf{Ind}_{\mathsf{L}(\rho)}^{l,A} : \forall l^{\mathsf{L}(\rho)}\, (A\,[l := \mathsf{nil}] \to \forall x^\rho \forall l^{\mathsf{L}(\rho)}\, (A \to A\,[l := x :: l]) \to A)$, denoting the *list induction* axiom for the formula A.

The proof terms of MA^ω are obtained by adding the clause

- $\bot^+ : \mathrm{F} \to \bot$, denoting the $\bot$ *introduction* axiom.

The proof terms of HA^ω are obtained by adding the clauses

- $\wedge^+_{A,B} : A \to B \to A \wedge B$, denoting the *conjunction introduction* axiom scheme;
- $\wedge^-_{A,B,C} : A \wedge B \to (A \to B \to C) \to C$, denoting the *conjunction elimination* axiom scheme for the formula C;
- $\exists^+_{x,A} : \forall x^\rho\, (A \to \exists x^\rho\, A)$, denoting the *existential introduction* axiom scheme;
- $\exists^-_{x,A,C} : \exists x^\rho\, A \to \forall x^\rho\, (A \to C) \to C$, denoting the *existential elimination* axiom scheme for the formula C. The scheme is subjected to the usual variable condition that $x \notin \mathsf{FV}(C)$.

Similarly to formulas, we extend the syntactic equality of proofs to identify terms participating in universal elimination rule instances if they have a common normal form. We also extend the reduction $\overset{\varepsilon}{\mapsto}$ to the proof rules involving universal formulas as follows:

$$(\lambda x^{\mathsf{I}}\, M^A)^{\forall x^{\mathsf{I}} A} \overset{\varepsilon}{\mapsto} M^A,$$
$$(M^{\forall x^{\mathsf{I}} A} \varepsilon^{\mathsf{I}})^{A[x:=\varepsilon]} \overset{\varepsilon}{\mapsto} M^A.$$

Sometimes it will be more convenient to use rules instead of the axiom schemes defined above. We will make use of some abbreviations, which define the introduction and elimination rules corresponding to some of the axiom schemes.

- $\langle M^A, N^B \rangle^{A \wedge B} := \wedge^+ MN$, denoting the *conjunction introduction* rule;
- $(M^{A \wedge B} \llcorner)^A := \wedge^- M(\lambda u \, \lambda v \, u)$, denoting the *left conjunction elimination* rule;
- $(M^{A \wedge B} \lrcorner)^B := \wedge^- M(\lambda u \, \lambda v \, v)$, denoting the *right conjunction elimination* rule;
- $\langle t^\rho, M^A \rangle^{\exists x^\rho A} := \exists^+ tM$, denoting the *existential introduction* rule.

The rest of the rules will be used with the usual syntax.

- $\exists^- M^{\exists x^\rho A}(\lambda x^\rho \, \lambda u^A \, N^C)$, denoting the *existential elimination* rule for the formula C discharging the assumption u and binding the variable x in the proof term N;
- $\mathcal{C} b^{\mathsf{B}} M^{A[b:=\mathsf{tt}]} N^{A[b:=\mathsf{ff}]}$, denoting the *boolean case distinction* rule for the formula A;
- $\mathsf{Ind}_{\mathsf{N}} n^{\mathsf{N}} M^{A[n:=0]}(\lambda n^{\mathsf{N}} \, \lambda u^A \, N^{A[n:=\mathsf{S}n]})$, denoting the *natural number induction* rule for the formula A, discharging the induction hypothesis u and binding the variable n in the proof term N;
- $\mathsf{Ind}_{\mathsf{L}(\rho)} l^{\mathsf{L}(\rho)} M^{A[l:=\mathsf{nil}]}(\lambda x^\rho \, \lambda l^{\mathsf{L}(\rho)} \, \lambda u^A \, N^{A[l:=x :: l]})$, denoting the *list induction* rule for the formula A, discharging the induction hypothesis u and binding the variables x and l in the proof term N.

The rules above are definable using the corresponding axioms, however, the converse is also true; the axiom schemes are definable in terms of the rules. Thus, we will use axioms and rules interchangeably when we argue on induction on the definition of a proof term, choosing whichever is most convenient.

In proof terms we have three types of variables: term (object) variables, assumption variables and (only in MA$^\omega$) the predicate variable $\bot$. We reserve the notations FV and BV for free and bound term variables and will use the notations FA and BA for free and bound assumption variables. We will call a proof in MA$^\omega$ $\bot$-free if all its formulas are $\bot$-free. Similarly to formulas, $\bot$-free proofs in MA$^\omega$ are exactly the proofs in NA$^\omega$.

Proof terms can be subjected to four kinds of substitutions: type, term, formula and proof substitutions. The application of all four kinds of substitutions is specified below.

Applying a **type** substitution to a proof term means applying it to:

- all formulas appearing in the proof term,
- the types of all terms appearing in universal elimination rule instances,
- the types of all variables bound by a universal introduction rule instance

Applying a **term** substitution to a proof term means applying it to:

- all formulas appearing in the proof term,
- all terms appearing in universal elimination rule instances.

Applying a **formula** substitution to a proof term means applying it to all formulas appearing in the proof term.

Applying a **proof** substitution is naturally defined for proofs using the general scheme described in Section 1.1.2.

Remark 1.39. In order to prove correctness of formula substitutions, we would need to show that the axiom $\bot^+$ preserves its validity under substitutions of the kind $[\bot := A]$ for an arbitrary formula A. We postpone this task for the next section.

Reductions can be defined for proof terms similarly to term reductions.

Definition 1.40 (Proof reductions [Pra71, Lei75]).

$$\begin{array}{llllll}
(\lambda u\, M)N & \mapsto & M\,[u := N]\,, & (\wedge^- M(\lambda u\, \lambda v\, N))T & \mapsto & \wedge^- M(\lambda u\, \lambda v\, NT),\ (*) \\
(\lambda x\, M)t & \mapsto & M\,[x := t]\,, & (\exists^- M(\lambda x\, \lambda u\, N))T & \mapsto & \exists^- M(\lambda x\, \lambda u\, NT),\ (*) \\
\mathsf{Ind}_{\mathsf{N}}\, 0\, M\, N & \mapsto & M, & \mathsf{Ind}_{\mathsf{N}}\, (\mathsf{S}n)\, M\, N & \mapsto & N\, n\, (\mathcal{R}_{\mathsf{N}}\, n\, M\, N), \\
\mathsf{Ind}_{\mathsf{L}(\rho)}\, \mathsf{nil}\, M\, N & \mapsto & M, & \mathsf{Ind}_{\mathsf{L}(\rho)}\, (n :: l)\, M\, N & \mapsto & N\, n\, l\, (\mathcal{R}_{\mathsf{L}(\rho)}\, l\, M\, N), \\
\mathcal{C}\, \mathsf{tt}\, M\, N & \mapsto & M, & (\mathcal{C}\, b\, M\, N)T & \mapsto & \mathcal{C}\, b\, (MT)\, (NT) \\
\mathcal{C}\, \mathsf{ff}\, M\, N & \mapsto & N, & \overset{\wedge}{\exists}{}^-(\mathcal{C}\, b\, M\, N) & \mapsto & \mathcal{C}\, b\, (\overset{\wedge}{\exists}{}^- M)\, (\overset{\wedge}{\exists}{}^- N) \\
\wedge^-(\wedge^+ M\, N)P & \mapsto & P\, M\, N & \overset{\wedge}{\exists}{}^-(\wedge^- M(\lambda u\, \lambda v\, N)) & \mapsto & \wedge^- M(\lambda u\, \lambda v\, \overset{\wedge}{\exists}{}^- N), \\
\exists^-(\exists^+ t\, M)N & \mapsto & N\, t\, M & \overset{\wedge}{\exists}{}^-(\exists^- M(\lambda x\, \lambda u\, N)) & \mapsto & \exists^- M(\lambda x\, \lambda u\, \overset{\wedge}{\exists}{}^- N), \\
\wedge^- M(\lambda u\, \lambda v\, T) & \mapsto & T,\ (*) & \exists^- M(\lambda x\, \lambda u\, T) & \mapsto & T,\ (*)
\end{array}$$

where $\overset{\wedge}{\exists}{}^-$ stands for $\wedge^-$ or $\exists^-$, and $(*)$ denotes the condition $u, v \notin \mathsf{FA}(T), x \notin \mathsf{FV}(T)$, with T standing for either an object or a proof term; and if $M \mapsto M'$, then

$$MN \mapsto M'N, \qquad NM \mapsto NM', \qquad \lambda u\, M \mapsto \lambda u\, M', \qquad \lambda x\, M \mapsto \lambda x\, M'.$$

Multiple-step reduction, set of reducts and normal form are defined as in Section 1.2.1. It is easy to establish that if $M^A \mapsto N^B$, then $A \overset{r}{=} B$, $\mathsf{FV}(N) \subseteq \mathsf{FV}(M)$ and $\mathsf{FA}(N) \subseteq \mathsf{FA}(M)$. It is well-known that HA^ω is strongly normalizing.

Theorem 1.41 (Strong normalization for HA^ω [Pra71, Lei75]). *For every proof term M^A, there is a proof term M_0^A, such that $M \overset{*}{\mapsto} M_0$ and M_0 is in normal form.*

As remarked in [Lei75], a simpler version of the argument applies to the subsystems NA^ω and MA^ω.

Theorem 1.42 (Subformula property [Pra71]). *Let M^A be a proof term in normal form and let $\mathsf{FA}(M) = \{u_i^{C_i}\}$. Then all formulas appearing in M are subformulas of the conclusion A or of some of the assumptions C_i.*

The three considered systems differ only in their formula language, and not in the derivation axioms and rules. Consequently, we can talk about derivability of formulas without explicitly specifying a system, as it will be determined by the language used in the formulas involved in the proof term. Moreover, the two theorems above imply that a proof term M, for which the conclusion and the assumptions are formulas in NA^ω can be carried out entirely in NA^ω when normalized. Therefore, the system which proves a certain formula will be entirely determined by the language of the conclusion and the assumptions.

Definition 1.43 (Derivability)**.** We say that a formula A is *derivable from assumptions* $\Gamma = \{C_i\}$ if there is a proof term M^A, such that $\mathsf{FA}(M) = \{u_i^{C_i}\}$ and denote $\mathsf{FA}(M) \vdash M : A$ or omitting proof terms $\Gamma \vdash A$. In case the proof M is closed, i.e., $\mathsf{FA}(M) = \emptyset$, we say that a formula A is *derivable* and denote $\vdash M : A$ or just A.

The fact that the notion of derivability is uniform across systems does not mean that the systems prove the same theorems. In particular, since the language of NA^ω is more restricted than the language of HA^ω, there are some *theorem schemata* involving an arbitrary formula A, which are valid if A ranges over NA^ω, but are invalid if we allow A to range over HA^ω. An example of this phenomenon is the stability principle described above.

1.4 System embeddings

In this section we will demonstrate how the systems MA^ω, NA^ω and HA^ω are embeddable in each other. We will start by proving some underlying principles, which should hold in any intuitionistic or classical arithmetical system.

Theorem 1.44 (Ex falso quodlibet)**.** $\vdash \mathrm{F} \to A$ *for any formula A in* NA^ω, MA^ω, *or* HA^ω.

Proof. The table below shows how to construct a proof $\mathsf{efq}_A^{\mathrm{F}\to A}$ by induction on the formula A. We assume that by induction hypothesis we have $M^{\mathrm{F}\to B}$ and $N^{\mathrm{F}\to C}$.

A	efq_A
$\mathrm{at}(t)$	$\lambda u^{\mathrm{F}} \mathcal{C}\, t\, u\, \mathsf{AxT}$
$\bot$	$\bot^+$
$B \to C$	$\lambda u^{\mathrm{F}} v^B N u$
$\forall x^\rho B$	$\lambda u^{\mathrm{F}} \lambda x^\rho M u$
$B \wedge C$	$\lambda u^{\mathrm{F}} \langle Mu, Nu \rangle$
$\exists x^\rho B$	$\lambda u^{\mathrm{F}} \langle \Box^\rho, M \rangle$

$\Box$

Remark 1.45. The theorem above is not the “real” ex falso quodlibet principle for MA^ω, because not F, but $\bot$ is the legitimate falsity for that system. As is to be expected, the ex falso quodlibet principle with $\bot$ for falsity does not hold in the minimal system MA^ω, because there is no way to prove $\bot \to \mathrm{F}$. However, the theorem shows that the axiom $\bot^+$ preserves its validity under arbitrary formula substitutions.

Theorem 1.46 (Stability). *$\vdash ((A \to \mathrm{F}) \to \mathrm{F}) \to A$ for any formula A in NA^ω.*

Proof. The table below shows how to construct a proof $\mathsf{stab}_A^{\neg\neg A \to A}$ by induction on the formula A. We assume that by induction hypothesis we have $M^{\neg\neg B \to B}$ and $N^{\neg\neg C \to C}$.

A	stab_A
$\mathrm{at}(t)$	$\mathcal{C}\, t\, (\lambda u^{\neg\neg \mathrm{T}} \mathsf{AxT})\, (\lambda u^{\neg\neg \mathrm{F}} u(\lambda v^{\mathrm{F}} v))$
$B \to C$	$\lambda u^{\neg\neg(B \to C)} \lambda x^B N\Big(\lambda v^{\neg C} u\big(\lambda w^{B \to C} v(wx)\big)\Big)$
$\forall x^\rho\, B$	$\lambda u^{\neg\neg \forall x\, B} \lambda x^\rho M\Big(\lambda v^{\neg B} u\big(\lambda w^{\forall x\, B} v(wx)\big)\Big)$

□

Remark 1.47. The stability principle is not valid for MA^ω: $((\bot \to \mathrm{F}) \to \mathrm{F}) \to \bot$ is not provable, since $\bot \to \mathrm{F}$ is not. Stability is also clearly not valid for HA^ω, because it does not hold at $\exists x\, B$.

Theorem 1.48 (Case distinction on terms). *For any formula A,*

$$\vdash \forall b\, \big((\mathrm{at}(b) \to A\,[b := \mathsf{tt}]) \to ((\mathrm{at}(b) \to \mathrm{F}) \to A\,[b := \mathsf{ff}]) \to A\big).$$

Proof. We define the proof $\mathsf{CD}_{b,A}$ by case distinction as follows:

$$\mathsf{CD}_{b,A} := \lambda b\, \mathcal{C}\, b\, (\lambda u^{\mathrm{T} \to A[b:=\mathsf{tt}]}\, \lambda v\, u\, \mathsf{AxT})\, \big(\lambda u\, \lambda v^{(\mathrm{F} \to \mathrm{F}) \to \bot}\, v(\lambda w^{\mathrm{F}}\, w)\big). \qquad \square$$

Definition 1.49 (System embedding). An *embedding* of a system $\mathcal{A}$ into a system $\mathcal{B}$ is a mapping Ω acting on formulas, such that

1. for every formula $A \in \mathcal{A}$ we have a formula $\Omega(A) \in \mathcal{B}$, such that $\mathsf{FV}(A) = \mathsf{FV}(\Omega(A))$ and $\mathsf{BV}(A) = \mathsf{BV}(\Omega(A))$;
2. $\Gamma \vdash A$ exactly when $\Omega(\Gamma) \vdash \Omega(A)$.

If instead 2. we have

2’. $\Gamma \vdash A$ implies $\Omega(\Gamma) \vdash \Omega(A)$,

we call Ω a *weak embedding.*

$\mathcal{A}$ is called *(weakly) embeddable* in $\mathcal{B}$ if there is a (weak) embedding of $\mathcal{A}$ into $\mathcal{B}$.

From the definition of the systems we immediately obtain some trivial embeddings.

Proposition 1.50. *The identity mapping embeds* NA^ω *into* HA^ω *and* MA^ω*;* $\mathrm{NA}^\omega_{\mathrm{at}}$ *into* NA^ω_0*; and* NA^ω_0 *into* HA^ω_0*.*

In fact, we will show that the systems $\mathrm{NA}^\omega_{\mathrm{at}}$, NA^ω_0 and HA^ω_0 possess the same logical strength by closing the loop with an embedding of HA^ω_0 into $\mathrm{NA}^\omega_{\mathrm{at}}$.

Lemma 1.51. There are closed terms $T_\to, T_\wedge : \mathsf{B} \Rightarrow \mathsf{B} \Rightarrow \mathsf{B}$ such that

1. $\vdash \mathrm{at}(T_\to xy) \leftrightarrow (\mathrm{at}(x) \to \mathrm{at}(y))$.
2. $\vdash \mathrm{at}(T_\wedge xy) \leftrightarrow (\mathrm{at}(x) \wedge \mathrm{at}(y))$.

Proof. Define $T_\to := \lambda x\, \lambda y\, \mathsf{Cases}\, x\, y\, \mathsf{tt}$ and $T_\wedge := \lambda x\, \lambda y\, \mathsf{Cases}\, x\, y\, \mathsf{ff}$. The following proof terms prove the two equivalences:

$$\begin{aligned}
\mathcal{P}_{\to l} &:= \mathcal{C}\, x\, (\lambda u^{\mathrm{T}\to\mathrm{at}(y)}\, u\mathsf{AxT})\, (\lambda u^{\mathrm{F}\to\mathrm{at}(y)}\, \mathsf{AxT}),\\
\mathcal{P}_{\to r} &:= \mathcal{C}\, x\, (\lambda u^{\mathrm{at}}(y)\, \lambda v^{\,\mathrm{T}} u)\, (\lambda u^{\mathrm{T}}\, \lambda v^{\mathrm{F}}\, \mathsf{efq}_{\mathrm{at}(y)} v),\\
\mathcal{P}_{\wedge l} &:= \mathcal{C}\, x\, (\lambda u^{\mathrm{T}\wedge\mathrm{at}(y)}\, u_{\lrcorner})\, (\lambda u^{\mathrm{F}\wedge\mathrm{at}(y)}\, u_{\llcorner}),\\
\mathcal{P}_{\wedge r} &:= \mathcal{C}\, x\, (\lambda u^{\mathrm{at}(y)}\, \langle \mathsf{AxT}, u\rangle)\, (\lambda u^{\mathrm{F}}\, \langle u, \mathsf{efq}_{\mathrm{at}(y)} u\rangle).
\end{aligned}$$ □

Corollary 1.52. *There is a closed term* $T_\neg : \mathsf{B} \Rightarrow \mathsf{B}$ *such that* $\vdash \mathrm{at}(T_\neg x) \leftrightarrow (\mathrm{at}(x) \to \mathrm{F})$*.*

Proof. Define $T_\neg := \lambda x\, (T_\to\, x\, \mathsf{ff})$. □

Definition 1.53 (Atomic translation)**.** For every formula C in HA^ω_0 we define the term C^{at} (the *atomic translation* of C) as follows:

$$\begin{aligned}
(\mathrm{at}(t))^{\mathrm{at}} &:= t,\\
(A \to B)^{\mathrm{at}} &:= T_\to(A^{\mathrm{at}})(B^{\mathrm{at}}),\\
(A \wedge B)^{\mathrm{at}} &:= T_\wedge(A^{\mathrm{at}})(B^{\mathrm{at}}).
\end{aligned}$$

Proposition 1.54. HA^ω_0 *is embeddable in* $\mathrm{NA}^\omega_{\mathrm{at}}$*.*

Proof. We define the mapping $\Omega(A) := \mathrm{at}(A^{\mathrm{at}})$ for any formula A in HA^ω_0. We prove that Ω is an embedding by induction on A. The atomic case is trivial and the other two cases are handled by Lemma 1.51. □

Corollary 1.55 (Case distinction on decidable formulas)**.** $\vdash (D \to A) \to ((D \to \mathrm{F}) \to A) \to A$ *for an arbitrary formula* A *and for any* $D \in \mathrm{HA}^\omega_0$*.*

Proof. Follows from Theorem 1.48 and Lemma 1.51 using $\mathsf{CD}_{b,A}(D^{\mathrm{at}})$ for b a fresh boolean variable, not appearing freely in A. We will denote the corresponding proof term as $\mathsf{CD}_{D,A}$. □

The identity embedding of NA^ω into HA^ω is very convenient. However, it is not as useful for embedding NA^ω into MA^ω. The reason is that we have different notions of falsity (and hence negation) in NA^ω and MA^ω. An honest embedding would translate the arithmetical falsity F from NA^ω into a formula, which is equivalent to $\bot$, the falsity in MA^ω. This can be achieved via the double negation translation [Tro73].

Definition 1.56 (Double negation translation)**.** We define the double negation translation $(\cdot)^{\neg\neg}$ as follows:

$$\begin{aligned}
(\mathrm{at}(t))^{\neg\neg} &:= (\mathrm{at}(t) \to \bot) \to \bot,\\
(A \to B)^{\neg\neg} &:= A^{\neg\neg} \to B^{\neg\neg},\\
(\forall x\, A)^{\neg\neg} &:= \forall x\, A^{\neg\neg}.
\end{aligned}$$

Clearly $\mathrm{F}^{\neg\neg} = (\mathrm{F} \to \bot) \to \bot$ is provably equivalent to $\bot$ via the the axiom $\bot^+$.

Remark 1.57. We would obtain a simpler translation by postulating $\mathrm{F}^{\neg\neg} := \bot$, as in the original Gödel-Gentzen negative translation. However, in our case it does not introduce a substantial simplification, so we prefer a uniform treatment of all atomic formulas.

Lemma 1.58. $\vdash A^{\neg\neg}[\bot := \mathrm{F}] \leftrightarrow A$.

Proof. Induction on A. The implication and universal quantification cases are trivial, so we need to prove the claim only for atomic formulas, i.e., $\vdash ((\mathrm{at}(t) \to \mathrm{F}) \to \mathrm{F}) \leftrightarrow \mathrm{at}(t)$. The direction "$\leftarrow$" is proved by $\lambda u^{\mathrm{at}(t)} \lambda v^{\neg \mathrm{at}(t)} vu$ and the direction "$\to$" is proved by $\mathsf{stab}_{\mathrm{at}(t)}$. □

Proposition 1.59. $(\cdot)^{\neg\neg}$ *is an embedding of* NA^ω *into* MA^ω*.*

Proof. First we prove that from any proof of $\Gamma \vdash A$ we can construct a proof of $\Gamma^{\neg\neg} \vdash A^{\neg\neg}$ by induction on the proof term in NA^ω. Implication and universal introduction and elimination rules translate to themselves, because $(\cdot)^{\neg\neg}$ goes through implications and universal quantifiers. The induction axioms $\mathcal{C}^{b,A}$, $\mathsf{Ind}_{\mathsf{N}}^{n,A}$ and $\mathsf{Ind}_{\mathsf{L}(\rho)}^{l,A}$ translate to $\mathcal{C}^{b,A^{\neg\neg}}$, $\mathsf{Ind}_{\mathsf{N}}^{n,A^{\neg\neg}}$ and $\mathsf{Ind}_{\mathsf{L}(\rho)}^{l,A^{\neg\neg}}$ respectively. Finally, the axiom AxT translates to $\lambda u^{\mathrm{T}\to\bot}(u\,\mathsf{AxT})$.

In order to show the converse, assume that $\Gamma^{\neg\neg} \vdash A^{\neg\neg}$. Substituting $[\bot := \mathrm{F}]$ and applying Lemma 1.58 we obtain $\Gamma \vdash A$. □

Proposition 1.60. MA^ω *is weakly embeddable into* NA^ω*.*

Proof. Define $\Omega(A) := A[\bot := \mathrm{F}]$. It is clear that for every proof term M^A we can obtain the proof $(M[\bot := \mathrm{F}])^{\Omega(A)}$. □

Definition 1.61 (Weakening translation). We define the *weakening translation* $(\cdot)^{\mathsf{w}}$ of formulas HA^ω to formulas in NA^ω as follows:

$$\begin{aligned}
(\mathrm{at}(t))^{\mathsf{w}} &:= \mathrm{at}(t),\\
(A \to B)^{\mathsf{w}} &:= A^{\mathsf{w}} \to B^{\mathsf{w}},\\
(\forall x\, A)^{\mathsf{w}} &:= \forall x\, A^{\mathsf{w}}\\
(A \wedge B)^{\mathsf{w}} &:= A^{\mathsf{w}} \mathbin{\tilde\wedge} B^{\mathsf{w}}\\
(\exists x\, A)^{\mathsf{w}} &:= \tilde\exists x\, A^{\mathsf{w}}.
\end{aligned}$$

Proposition 1.62. $(\cdot)^{\mathsf{w}}$ *is a weak embedding of* HA^ω *into* NA^ω.

Proof. Let A be a formula in HA^ω and assume that we have a proof term M^A. We argue by induction on M and we need to consider only the axioms of HA^ω, which are not present in NA^ω. The table below summarizes how a proof d^{w} is constructed for the weak translation each of the axioms d^D.

$d : D$	d^{w}
$\wedge^+_{A,B}$	$\lambda u^{A^{\mathsf{w}}} \lambda v^{B^{\mathsf{w}}} \lambda w^{A^{\mathsf{w}} \to B^{\mathsf{w}} \to \mathrm{F}} wuv$
$\wedge^-_{A,B,C}$	$\lambda u^{A^{\mathsf{w}} \tilde\wedge B^{\mathsf{w}}} \lambda v^{A^{\mathsf{w}} \to B^{\mathsf{w}} \to C^{\mathsf{w}}} \mathsf{stab}_{C^{\mathsf{w}}}(\lambda w^{\neg C^{\mathsf{w}}} u(\lambda z_0^{A^{\mathsf{w}}} \lambda z_1^{B^{\mathsf{w}}} w(vz_0z_1)))$
$\exists^+_{x,A}$	$\lambda x\, \lambda u^{A^{\mathsf{w}}} \lambda v^{\forall x \neg A^{\mathsf{w}}} vxu$
$\exists^-_{x,A,C}$	$\lambda u^{\tilde\exists x\, A^{\mathsf{w}}} \lambda v^{\forall x\, (A^{\mathsf{w}} \to C^{\mathsf{w}})} \mathsf{stab}_{C^{\mathsf{w}}}(\lambda w^{\neg C^{\mathsf{w}}} u(\lambda x\, \lambda z^{A^{\mathsf{w}}} w(vxz)))$

□

In the text we will use the weak translation for notational convenience, as well as the proof terms $(\wedge^+_{A,B})^{\mathsf{w}}$, $(\wedge^-_{A,B,C})^{\mathsf{w}}$, $(\exists^+_{x,A})^{\mathsf{w}}$ and $(\exists^-_{x,A,C})^{\mathsf{w}}$.

It is important to note that the weak connectives $\tilde\wedge$ and $\tilde\exists$ should be used with caution, as their careless application can introduce an unnecessary number of negations in the resulting formula, which will inevitably lead to complications in the proofs. We will thus define special shortcut notations for repeated use of $\tilde\wedge$ and $\tilde\exists$ as follows:

$$\begin{aligned}
A \mathbin{\tilde\wedge} B \mathbin{\tilde\wedge} C &:= \neg(A \to B \to C \to \bot),\\
\tilde\exists x, y\, A &:= \neg\forall x\, \forall y\, (\neg A),\\
\tilde\exists x\, (A \mathbin{\tilde\wedge} B) &:= \neg\forall x\, (A \to B \to \bot).
\end{aligned}$$

CHAPTER

TWO

PROGRAM EXTRACTION FROM PROOFS

In this chapter three methods for extraction of programs from proofs will be presented. The first of them is modified realisability [Kre59], which is a formalisation of the Curry-Howard correspondence, transforming constructive proofs into functional programs. The next method, refined A-translation [BBS02] is based on modified realisability and extracts programs from proofs in minimal logic. The chapter will conclude with Gödel's Dialectica interpretation, which is able to extract higher-order programs from classical proofs.

2.1 Modified realisability

The first notion of realisability was developed by Kleene [Kle45] with the aim to provide a computational model of the Brower-Heyting-Kolmogorov interpretation of constructive logic. In this interpretation formulas in HA^ω can be viewed as problems, which require a solution in the following manner:

- atomic formulas $\mathrm{at}(t)$ do not require a solution, they state facts;
- an implication $A \to B$ requires a function, which maps every solution of A to a solution of B;
- a conjunction $A \wedge B$ requires a pair of solutions for A and B respectively;
- a universal formula $\forall x\, A$ requires a function, which maps every possible value v of the variable x to a solution of $A\,[x := v]$;
- an existential formula $\exists x\, A$ requires a pair of solutions: a value v for x and a solution of $A\,[x := v]$.

Note that the existential quantifier plays a central role: a formula without any existential quantifiers cannot require solutions. A constructive proof of a formula A

should achieve two goals simultaneously: construct a solution for A and prove that the solution is correct.

The realisability interpretation formalises the above inductive definition via a *realisability predicate* — a relation between computations and formulas, which defines when a given computation solves a certain formula. Kleene suggested to use program codes to denote computations and even though this was a correct model of constructive logic [Nel47], it turned out to be an incomplete one [Ros53]. A better approach was suggested later by Kreisel [Kre59]: computations should be modelled as (typed) λ-terms. This variant was called "modified realisability" and became popular for extracting programs from constructive proofs.

2.1.1 Definition

We will define the following components of the modified realisability interpretation:

1. $\tau^\circ(A)$ — the type of the required solutions of the formula A;
2. $t\ \mathbf{r}\ A$ — the realisability predicate relating a formula A and a term $t^{\tau^\circ(A)}$;
3. $[\![M]\!]^\circ$ — the extracted term from a proof term M.

Definition 2.1 (Realisability computational type)**.** For every formula A in HA$^\omega$ we define its computational type $\tau^\circ(A)$ inductively as follows:

$$\begin{aligned}
\tau^\circ(\mathrm{at}(t)) &:= \mathsf{I},\\
\tau^\circ(B \to C) &:= \tau^\circ(B) \Rightarrow \tau^\circ(C),\\
\tau^\circ(B \wedge C) &:= \tau^\circ(B) \times \tau^\circ(C),\\
\tau^\circ(\forall x^\rho\, B) &:= \rho \Rightarrow \tau^\circ(B),\\
\tau^\circ(\exists x^\rho\, B) &:= \rho \times \tau^\circ(B).
\end{aligned}$$

If $\tau^\circ(A) \neq \mathsf{I}$, we call A *computationally relevant*. If $\tau^\circ(A) = \mathsf{I}$, we call A *computationally irrelevant*.

Proposition 2.2. *Let A be a formula in* HA$^\omega$*. Then A is computationally relevant iff there is a strictly positive occurrence of an existential formula in A.*

Proof. Induction on the definition of A.

Case at(t). Clear.

Case $B \to C$. By definition, $\tau^\circ(B \to C) \neq \mathsf{I}$ exactly when $\tau^\circ(C) \neq \mathsf{I}$. On the other hand, an existential formula occurs strictly positively in $B \to C$ exactly when it occurs strictly positively in C and we finish the proof by using the induction hypothesis.

Case $B \wedge C$. By definition, $\tau^\circ(B \wedge C) = \mathsf{I}$ exactly when $\tau^\circ(B) = \tau^\circ(C) = \mathsf{I}$. On the other hand, an existential formula occurs strictly positively in $B \wedge C$ exactly when it occurs strictly positively in B or in C and we finish the proof by using the induction hypothesis.

Case $\forall x\, B$. By definition, $\tau^\circ(\forall x\, B) \neq \mathsf{I}$ exactly when $\tau^\circ(B) \neq \mathsf{I}$ and we finish the proof by using the induction hypothesis.

Case $\exists x\, B$. Clear, because by definition, $\tau^\circ(\exists x\, B)$ cannot collapse to I. $\square$

Proposition 2.2 implies that all formulas in NA^ω are computationally irrelevant, which shows that modified realisability is non-trivially applicable only for constructive proofs.

Definition 2.3 (Realisability predicate)**.** We define inductively the value of the realisability predicate at a formula A in HA^ω and a term $t : \tau^\circ(A)$, to be read *"A is realised by t"*, as follows:

$$\begin{aligned}
\varepsilon\ \mathbf{r}\ \mathrm{at}(t) &:= \mathrm{at}(t),\\
t\ \mathbf{r}\ (B \to C) &:= \forall x^{\tau^\circ(B)}\, (x\ \mathbf{r}\ B \to tx\ \mathbf{r}\ C), \text{ for a fresh variable } x,\\
t\ \mathbf{r}\ (B \wedge C) &:= (t_\llcorner\ \mathbf{r}\ B)\,\tilde{\wedge}(t_\lrcorner\ \mathbf{r}\ C),\\
t\ \mathbf{r}\ (\forall x^\rho\, B) &:= \forall x\, (tx\ \mathbf{r}\ B),\\
t\ \mathbf{r}\ (\exists x^\rho\, B) &:= t_\llcorner\ \mathbf{r}\ B\, [x := t_\lrcorner].
\end{aligned}$$

The following proposition is easily proved by induction on A.

Proposition 2.4. *For every formula A in* HA^ω *and term t:*

1. $\tau^\circ(A\,[x := t]) = \tau^\circ(A)$;
2. $\mathsf{FV}(t\ \boldsymbol{r}\ A) \subseteq \mathsf{FV}(A) \cup \mathsf{FV}(t)$.
3. *$t\ \boldsymbol{r}\ A$ is a formula in* NA^ω*;*

In fact, all formulas in NA^ω are invariant with respect to realisability.

Proposition 2.5 (NA^ω realisability invariance)**.** *$\varepsilon\ \boldsymbol{r}\ A = A$ for every formula A in* NA^ω.

Proof. Induction on A.

Case $\mathrm{at}(t)$. By definition.

Case $B \to C$. $\varepsilon\ \mathbf{r}\ (B \to C) = \forall x^{\mathsf{I}}\, (x\ \mathbf{r}\ B \to \varepsilon x\ \mathbf{r}\ C) = B \to C$ by induction hypothesis and the I convention.

Case $\forall x\, B$. $\varepsilon\ \mathbf{r}\ (\forall x\, B) = \forall x\, (\varepsilon x\ \mathbf{r}\ B) = \forall x\, B$ by induction hypothesis and the I convention. $\square$

2.1.2 Soundness

In this section we will prove soundness of the modified realisability interpretation.

Theorem 2.6 (Soundness of modified realisability)**.** *Let $\mathcal{P}^A$ be a proof term in* HA^ω *with assumptions* $\mathsf{FA}(\mathcal{P}) = \{u_i : C_i\}_{i\geq 1}$*. Let us have a set of fresh variables* $X = \{x_i : \tau^\circ(C_i)\}$ *and a set of fresh assumption variables* $V = \{v_i : (x_i \mathbf{\,r\,} C_i)\}$*, each one uniquely associated with each of the assumption variables* u_i*. Then there is a term* $[\![\mathcal{P}]\!]^\circ : \tau^\circ(A)$ *and a proof term* $\overline{\mathcal{P}} : [\![\mathcal{P}]\!]^\circ \mathbf{\,r\,} A$*, such that* $\mathsf{FA}(\overline{\mathcal{P}}) \subseteq V$ *and* $\mathsf{FV}([\![\mathcal{P}]\!]^\circ) \subseteq \mathsf{FV}(\overline{\mathcal{P}}) \subseteq \mathsf{FV}(\mathcal{P}) \cup X$*.*

Proof. We define $[\![\mathcal{P}]\!]^\circ$ and $\overline{\mathcal{P}}$ by induction on the proof term $\mathcal{P}$.

Case AxT. Set $[\![\mathcal{P}]\!]^\circ := \varepsilon$ and $\overline{\mathcal{P}} := \mathsf{AxT}$.

Case $u_1^{C_1}$. Set $[\![\mathcal{P}]\!]^\circ := x_1$ and $\overline{\mathcal{P}} := v_1$.

Case $\lambda u_0^B\, M^C$. By induction hypothesis we have a proof term $\overline{M} : [\![M]\!]^\circ \mathbf{\,r\,} C$ with assumptions v_i for $i \geq 0$. Set $[\![\mathcal{P}]\!]^\circ := \lambda x_0\, [\![M]\!]^\circ$ and $\overline{\mathcal{P}} := \lambda x_0\, \lambda v_0\, \overline{M}$. The universal introduction in $\overline{\mathcal{P}}$ is correct, because x_0 appears only in the assumption v_0. The variable conditions are satisfied, because $\mathsf{FA}(\overline{\mathcal{P}}) = \mathsf{FA}(\overline{M}) \setminus \{v_0\}$ and $\mathsf{FV}(\overline{\mathcal{P}}) = \mathsf{FA}(\overline{M}) \setminus \{x_0\}$.

Case $M^{B\to A}N^B$. By induction hypothesis we have proof terms $\overline{N} : [\![N]\!]^\circ \mathbf{\,r\,} B$ and $\overline{M} : \forall x^{\tau^\circ(B)}\, (x \mathbf{\,r\,} B \to [\![M]\!]^\circ x \mathbf{\,r\,} A)$ with $\mathsf{FA}(\overline{M}) \cup \mathsf{FA}(\overline{N}) \subseteq V$. Set $[\![\mathcal{P}]\!]^\circ := [\![M]\!]^\circ [\![N]\!]^\circ$ and $\overline{\mathcal{P}} := \overline{M}[\![N]\!]^\circ\overline{N}$. The variable conditions are satisfied, because $\mathsf{FA}(\overline{\mathcal{P}}) = \mathsf{FA}(\overline{M}) \cup \mathsf{FA}(\overline{N})$ and $\mathsf{FV}(\overline{\mathcal{P}}) = \mathsf{FV}(\overline{M}) \cup \mathsf{FV}(\overline{N})$ (since $\mathsf{FV}(\overline{N}) \supseteq \mathsf{FV}([\![N]\!]^\circ)$).

Case $\lambda x^\rho\, M^B$. By induction hypothesis we have a proof term $\overline{M} : [\![M]\!]^\circ \mathbf{\,r\,} B$. Set $[\![\mathcal{P}]\!]^\circ := \lambda x\, [\![M]\!]^\circ$ and $\overline{\mathcal{P}} := \lambda x\, \overline{M}$. The universal introduction in $\overline{\mathcal{P}}$ is correct, because it is correct in $\mathcal{P}$ and by induction hypothesis $\mathsf{FA}(\overline{M}) \subseteq V$ and $\mathsf{FV}[V] \subseteq \mathsf{FV}[\mathsf{FA}(M)] \cup X$. The variable conditions are satisfied, because $\mathsf{FA}(\overline{\mathcal{P}}) = \mathsf{FA}(\overline{M})$ and $\mathsf{FV}(\overline{\mathcal{P}}) = \mathsf{FV}(\overline{M}) \setminus \{x\}$.

Case $M^{\forall x\, B}t$. By induction hypothesis we have a proof term $\overline{M} : \forall x\, ([\![M]\!]^\circ x \mathbf{\,r\,} B)$. Set $[\![\mathcal{P}]\!]^\circ := [\![M]\!]^\circ t$ and $\overline{\mathcal{P}} := \overline{M}t$. The variable conditions are satisfied, because $\mathsf{FA}(\overline{\mathcal{P}}) = \mathsf{FA}(\overline{M})$ and $\mathsf{FV}(\overline{\mathcal{P}}) = \mathsf{FV}(\overline{M}) \cup \mathsf{FV}(t) \subseteq \mathsf{FV}(\mathcal{P}) \cup X$.

Case $\wedge^+_{B,C}$. We define $[\![\mathcal{P}]\!]^\circ := \mathsf{Pair}_{\tau^\circ(B),\tau^\circ(C)}$ and

$$\overline{\mathcal{P}} := \lambda x_B\, \lambda v_B\, \lambda x_C\, \lambda v_C\, \lambda u^{(x_B \mathbf{\,r\,} B)\to(x_C \mathbf{\,r\,} C)\to\mathrm{F}} u v_B v_C,$$

where $x_\phi : \tau^\circ(\phi)$ and $v_\phi : (x_\phi \mathbf{\,r\,} \phi)$ for $\phi \in \{B, C\}$.

Case $\wedge^-_{B,C,D}$. We define $[\![\mathcal{P}]\!]^\circ := \lambda x\, \mathsf{Split}^{\tau^\circ(D)}_{\tau^\circ(B),\tau^\circ(C)}\, \langle x_\llcorner, x_\lrcorner\rangle$[1]. Let us denote $G := B \wedge C$ and $H := B \to C \to D$. Then we can define

$$\overline{\mathcal{P}} := \lambda x_G\, \lambda v_G\, \lambda x_H\, \lambda v_H\, (\wedge^-)^{\mathsf{w}} v_G \big(\lambda v_B\, \lambda v_C\, v_H(x_G\llcorner) v_B (x_G\lrcorner) v_C\big),$$

[1]If we had admitted the η-expansion $x \mapsto \langle x_\llcorner, x_\lrcorner\rangle$ and had considered η-long normal forms, then we could have defined $[\![\mathcal{P}]\!]^\circ := \mathsf{Split}^{\tau^\circ(D)}_{\tau^\circ(B),\tau^\circ(C)}$

where $x_\phi : \tau^\circ(\phi)$ and $v_\phi : (x_\phi \mathbf{\ r\ } \phi)$ for $\phi \in \{B, C, G, H\}$, since by definition

$$\begin{aligned}(x_G \mathbf{\ r\ } G) &= (x_{G\llcorner} \mathbf{\ r\ } B \to x_{G\lrcorner} \mathbf{\ r\ } C \to \mathrm{F}) \to \mathrm{F},\\ (x_H \mathbf{\ r\ } H) &= \forall x_B\,(x_B \mathbf{\ r\ } B \to \forall x_C\,(x_C \mathbf{\ r\ } C \to x_H x_B x_C \mathbf{\ r\ } D)).\end{aligned}$$

Case $\exists^+_{x^\rho, B}$. We define $[\![\mathcal{P}]\!]^\circ := \mathsf{Pair}_{\rho,\tau^\circ(B)}$. Since by definition $(\mathsf{Pair}\,x\,x_B) \mathbf{\ r\ } \exists x\, B = x_B \mathbf{\ r\ } B$, we can directly define $\overline{\mathcal{P}} := \lambda x\, \lambda x_B\, \lambda v_B\, v_B$, where $x_B : \tau^\circ(B)$ and $v_B : x_B \mathbf{\ r\ } B$.

Case $\exists^-_{x^\rho, B, C}$. We define $[\![\mathcal{P}]\!]^\circ := \lambda x\, \mathsf{Split}^{\tau^\circ(C)}_{\rho,\tau^\circ(B)} \langle x_\llcorner, x_\lrcorner \rangle$. Let us denote $G := \exists x\, B$ and $H := \forall x\,(B \to C)$. Then we can define

$$\overline{\mathcal{P}} := \lambda x_G\, \lambda v_G\, \lambda x_H\, \lambda v_H\, v_H(x_{G\llcorner})(x_{G\lrcorner}) v_G,$$

where $x_\phi : \tau^\circ(\phi)$ and $v_\phi : (x_\phi \mathbf{\ r\ } \phi)$ for $\phi \in \{B, C, G, H\}$, since by definition

$$\begin{aligned}(x_G \mathbf{\ r\ } G) &= (x_{G\lrcorner}) \mathbf{\ r\ } B\,[x := x_{G\llcorner}],\\ (x_H \mathbf{\ r\ } H) &= \forall x\, \forall x_B\,(x_B \mathbf{\ r\ } B \to x_H x x_B \mathbf{\ r\ } C).\end{aligned}$$

Case $\mathcal{C}^{b,C}$. We define $[\![\mathcal{P}]\!]^\circ := \mathsf{Cases}^{\tau^\circ(C)}$ and

$$\overline{\mathcal{P}} := \lambda b\, \lambda x_{\mathsf{tt}}\, \lambda v_{\mathsf{tt}}\, \lambda x_{\mathsf{ff}}\, \lambda v_{\mathsf{ff}}\, \mathcal{C}^{b, \mathsf{Cases}\, b x_{\mathsf{tt}} x_{\mathsf{ff}} \mathbf{\ r\ } C} b v_{\mathsf{tt}} v_{\mathsf{ff}},$$

where $x_i : \tau^\circ(C)$ and $v_i : (x_i \mathbf{\ r\ } C\,[b := i])$ for $i \in \{\mathsf{tt}, \mathsf{ff}\}$.

Case $\mathsf{Ind}^{n,C}_{\mathsf{N}}$. We define $[\![\mathcal{P}]\!]^\circ := \mathcal{R}^{\tau^\circ(C)}_{\mathsf{N}}$. Let us denote $C_0 := C\,[n := 0]$ and $C_{\mathsf{S}} := \forall n\,(C \to C\,[n := \mathsf{S}n])$. Then we can define

$$\overline{\mathcal{P}} := \lambda n\, \lambda x_0\, \lambda v_0\, \lambda x_{\mathsf{S}}\, \lambda v_{\mathsf{S}}\, \mathsf{Ind}^{n, \mathcal{R}_{\mathsf{N}} n x_0 x_{\mathsf{S}} \mathbf{\ r\ } C}_{\mathsf{N}} n v_0 (\lambda n\, v_{\mathsf{S}} n (\mathcal{R}_{\mathsf{N}} n x_0 x_{\mathsf{S}}))),$$

where $x_i : \tau^\circ(C_i)$ and $v_i : (x_i \mathbf{\ r\ } C_i)$ for $i \in \{0, \mathsf{S}\}$, since by definition

$$(x_{\mathsf{S}} \mathbf{\ r\ } C_{\mathsf{S}}) = \forall n\, \forall x_n^{\tau^\circ(C)}\,(x_n \mathbf{\ r\ } C \to x_{\mathsf{S}} n x_n \mathbf{\ r\ } C\,[n := \mathsf{S}n]).$$

Case $\mathsf{Ind}^{l,C}_{\mathsf{L}(\rho)}$. We define $[\![\mathcal{P}]\!]^\circ := \mathcal{R}^{\tau^\circ(C)}_{\mathsf{L}(\rho)}$. Let us denote $C_{\mathsf{nil}} := C\,[l := \mathsf{nil}]$ and $C_{::} := \forall x\, \forall x\,(C \to C\,[l := x :: l])$. Then we can define

$$\overline{\mathcal{P}} := \lambda n\, \lambda x_{\mathsf{nil}}\, \lambda v_{\mathsf{nil}}\, \lambda x_{::}\, \lambda v_{::}\, \mathsf{Ind}^{l, \mathcal{R}_{\mathsf{L}(\rho)} l x_{\mathsf{nil}} x_{::} \mathbf{\ r\ } C}_{\mathsf{L}(\rho)} l v_{\mathsf{nil}} (\lambda x\, \lambda l\, v_{::} x l (\mathcal{R}_{\mathsf{L}(\rho)} l x_{\mathsf{nil}} x_{::}))),$$

where $x_i : \tau^\circ(C_i)$ and $v_i : (x_i \mathbf{\ r\ } C_i)$ for $i \in \{\mathsf{nil}, ::\}$, since by definition

$$(x_{::} \mathbf{\ r\ } C_{::}) = \forall x\, \forall l\, \forall x_l^{\tau^\circ(C)}\,(x_l \mathbf{\ r\ } C \to x_{::} x l x_l \mathbf{\ r\ } C\,[l := x :: l]). \qquad \square$$

Corollary 2.7 (Program extraction via modified realisability)**.** *Let A be a formula in* NA^ω *and let* $\mathrm{HA}^\omega \vdash \forall x^\rho \, \exists y^\sigma \, A$*. Then there is a term* $t^{\rho \Rightarrow \sigma} : \tau^\circ(A)$*, such that* $\vdash \forall x^\rho \, A \, [y := tx]$*.*

Proof. By definition and Proposition 2.5 we have

$$\begin{aligned}
&\tau^\circ(\forall x \, \exists y \, A) = \rho \Rightarrow \sigma, \\
&t \; \mathbf{r} \; (\forall x \, \exists y \, A) = \forall x \, (tx \; \mathbf{r} \; \exists y \, A) = \forall x \, (\varepsilon \; \mathbf{r} \; A \, [y := tx]) = \forall x^\rho \, A \, [y := tx] \, .
\end{aligned}$$

Assuming that we have a proof $\mathcal{P}$ of $\forall x \, \exists y \, A$, by Theorem 2.6 we obtain $t := [\![\mathcal{P}]\!]^\circ$ and $\mathsf{FV}(t) \subseteq \mathsf{FV}(\overline{\mathcal{P}}) \subseteq \mathsf{FV}(\mathcal{P})$. □

2.2 Refined A-translation

Modified realisability is a complete and satisfactory method to obtain correct programs from constructive proofs. The realisability translation is quite straightforward, because by design constructive programs contain explicit witness constructions. However, classical principles are utilised to provide indirect proofs, it is not very easy to obtain a program, which in some way reflects the computational content of a certain proof.

A proof of weak completeness of intuitionistic arithmetic [Kre62] implies that Π^0_2 formulas that are provable in classical logic are also provable in intuitionistic logic. As a consequence, for every provably total recursive function we should be able to find a program which computes it. One way to obtain such a program is to apply Gödel's functional interpretation, which is discussed in the next section, to Π^0_2 formulas. However, Friedman [Fri78] and Dragalin [Dra80] independently suggested an easier way to prove the equiderivability of Π^0_2 formulas in classical and intuitionistic logic. This results gives rise to a method for obtaining a program from a non-constructive totality proof. A variant of the method could be summarised in the following three steps:

1. Start with a proof of a formula $\forall x \, \exists y \, \mathrm{at}(r)$ in classical logic and convert it to a proof of falsity (i.e., a proof from contradiction) in minimal logic via some negative embedding.
2. Since falsity plays no special role in minimal logic, we can soundly substitute any formula for it. We choose to substitute falsity with the formula $\exists y \, \mathrm{at}(r)$. As a result we obtain a proof of $\forall x \, \exists y \, \mathrm{at}(r)$, which is now constructive. This step is usually referred to as "A-translation".
3. We conclude by applying modified realisability to the translated proof in order to obtain a program r, for which $\forall x \, \mathrm{at}(t \, [y := rx])$.

A formal statement of the method in our setting is captured by the following

Theorem 2.8 (A-translation)**.** *Let* $\vdash \forall x^\rho \tilde{\exists} y^\sigma B$*, where* B *is a formula in* HA_0^ω *with* $\mathsf{FV}(B) \subseteq \{x, y\}$*. Then there is a closed term* $t^{\rho \Rightarrow \sigma}$ *such that* $\vdash \forall x\, B\, [y := tx]$*.*

Proof. Let A be a fixed formula. We define the A-translation of an arbitrary formula H as $H^A := H^{\neg\neg}[\bot := A]$. Let $b := B^{\mathrm{at}}, C := (\tilde{\exists} y\, \mathrm{at}(b))^A$. By the embedding properties of $(\cdot)^{\mathrm{at}}$ and $(\cdot)^{\neg\neg}$, we have a proof M of the formula

$$\begin{aligned} C &= (\forall y\, (\mathrm{at}(b) \to \mathrm{F}) \to \mathrm{F})^{\neg\neg}\, [\bot := A] \\ &= \forall y\, \big(((\mathrm{at}(b) \to A) \to A) \to (\mathrm{F} \to A) \to A)\big) \to (\mathrm{F} \to A) \to A. \end{aligned}$$

Now let $A := \exists y\, \mathrm{at}(b)$. We define a proof term $\mathcal{P}^{\forall x\, A}$ as follows:

$$\mathcal{P} := \lambda x\, M\big(\lambda y\, \lambda u^{\,(\mathrm{at}(b) \to A) \to A} \lambda v^{\mathrm{F} \to A}\, u(\exists^+_{y,A} y)\big)\, (\mathsf{efq}_A),$$

where $\exists^+_{y,A} : \forall y\, (\mathrm{at}(b) \to A)$. Finally, using Corollary 2.7 we extract the term $t := [\![\mathcal{P}]\!]^\circ$ such that $\vdash \forall x\, \mathrm{at}(b\, [y := tx])$, which by the embedding properties of $(\cdot)^{\mathrm{at}}$ implies $\vdash \forall x\, B\, [y := tx]$. $\square$

Remark 2.9. The original A-translation [Fri78, Dra80, Lei85] consisted of translating every atomic formula B to $B \vee A$. In our context $\vee$ is not a base connective, so such a translation would amount to replacing every atomic formula $\mathrm{at}(r)$ for example with $(\mathrm{at}(r))^{\mathcal{A}} := \exists b\, \big((b = \mathsf{tt} \to \mathrm{at}(r)) \wedge (b = \mathsf{ff} \to A)\big)$. It is easy to see that $(\mathrm{at}(r))^{\mathcal{A}}$ is in fact equivalent to the syntactically simpler translation $(\mathrm{at}(r))^A = (\mathrm{at}(r) \to A) \to A$.

The rather direct approach presented above has two major disadvantages. The first one is the restriction to Π^0_2 formulas only. This problem was addressed by Leivant, where the result is generalised to a wider syntactic class of *internally isolating* formulas [Lei85]. However, the second drawback still remains: the translation does not allow control over the computational content. In particular, the $(\cdot)^{\neg\neg}$ translation artificially pushes computational content in all atomic formulas, even where it is not really required. For modified realisability we could discard whole lemmas as computationally irrelevant, but now every A-translated proof has computational content. Consequently, we obtain an overcomplicated high order program, in which many values are computed only to be discarded on a later step.

Clearly, such a situation is not desirable from a practical point of view. This problem was treated by Buchholz, Berger and Schwichtenberg in [BBS02], where they suggested an optimisation, which became known as the *refined A-translation*. The idea presents a viewpoint shift: instead of blindly (and inefficiently) embedding classical proofs in minimal logic, we can attempt to find a suitable proof directly in minimal logic that can be translated to a constructive existence proof of the desired formula. In particular, let us assume that we would like to find a witness for a derivable formula $\vec{D} \to \tilde{\exists} y\, G$ in NA^ω, i.e., a term t such that $\vec{D} \to G\,[y := t]$. In order to do this, we can search for suitable *intermediate formulas* $\vec{D}'$ and G' in MA^ω for which

(1) we can find a proof M' of $\vec{D}' \to \forall y\,(G' \to \bot) \to \bot$, and
(2) we can translate M' to a proof M of $\vec{D} \to \forall y\,(G \to \bot) \to \bot$.

Since $\vec{D}, G \in \mathrm{NA}^\omega$, in (2) there are only two occurrences of $\bot$, and thus from the proof $M\,[\bot := \exists y\, G]$ we can obtain a proof of $\vec{D} \to \exists y\, G$, already in HA^ω.

By Theorem 2.8 we are sure that for the Π^0_2 case there is at least one solution to (1) and (2), namely taking $D_i' := (D_i)^{\neg\neg}$ and $G' := G^{\neg\neg}$; we can obtain the proof M' as in (1) from the original NA^ω proof of $\vec{D} \to \tilde{\exists} y\, G$ via the properties of the $(\cdot)^{\neg\neg}$ translation (Proposition 1.59) and M as in (2) is obtained via Lemma 1.58 by substituting $[\bot := \mathrm{F}]$ in $\vec{D}'$ and G'. In order to obtain simpler extracted terms t, our goal is to find $\vec{D}'$ and G' satisfying (1) and (2) such that M is as direct as possible.

The paper [BBS02] gives no algorithm to automatically find solutions to (1) and (2) above; instead, it describes classes of formulas D' and G' for which (2) is obtained automatically from (1). The method suggests to consider formulas $\vec{D}'$ and G' for which $D_i'\,[\bot := \mathrm{F}] = D_i$ and $G'\,[\bot := \mathrm{F}] = G$, where D_i' is a *definite* formula and G' is a *goal* formula. If we manage to prove (1) for such formulas, we can immediately generate a proof as in (2) and hence extract a witness.

We will present the refined A-translation technique below. For notational convenience, we extend the definition of realisability computational type to MA^ω by defining $\tau^\circ(\bot) := \alpha_\bot$, for $\alpha_\bot$ a fixed type variable. Let us define

$$\mathcal{R} := \{A \in \mathrm{MA}^\omega : \tau^\circ(A) \neq \mathsf{I}\}\,, \text{ the class of computationally relevant formulas,}$$
$$\mathcal{I} := \{A \in \mathrm{MA}^\omega : \tau^\circ(A) = \mathsf{I}\}\,, \text{ the class of computationally irrelevant formulas.}$$

Definition 2.10 (Definite and goal formulas [SW10])**.** We define the classes $\mathcal{D}$ of *definite* formulas and $\mathcal{G}$ of *goal* formulas in MA^ω by simultaneous induction.

1. $\bot, \mathrm{at}(t) \in \mathcal{D} \cap \mathcal{G}$.

Let $D \in \mathcal{D}$ and $G \in \mathcal{G}$, then

2. $D \to G \in \mathcal{G}$, if $D \in \mathrm{MA}_0^\omega \cup \mathcal{R}$ or $G \in \mathcal{I}$;
3. $G \to D \in \mathcal{D}$, if $G \in \mathcal{I}$ or $D \in \mathcal{R}$;
4. $\forall x\, G \in \mathcal{G}$, if $G \in \mathcal{I}$;
5. $\forall x\, D \in \mathcal{D}$.

Remark 2.11. Note that the definition above implies that every $\bot$-free formula is both definite and goal. This was not the case in the original definition in [BBS02], which had the weaker clause

2'. $D \to G \in \mathcal{G}$, if $D \in \mathrm{MA}_0^\omega \cup \mathcal{R}$.

This clause was revised in [SW10].

In the following, let us denote $A^F := A\,[\bot := \mathrm{F}]$.

Lemma 2.12 ([BBS02, SW10])**.** Let $D \in \mathcal{D}$ and $G \in \mathcal{G}$. Then the following formulas are provable in MA^ω:

(i) $D^F \to D$,
(ii) $G \to (G^F \to \bot) \to \bot$,
(iii) $((D^F \to \mathrm{F}) \to \bot) \to D$ for $D \in \mathcal{R}$,
(iv) $G \to G^F$ for $G \in \mathcal{I}$,

Proof. We will prove the claims (i)–(iv) by simultaneous induction on the definition of the formula involved.

Case $\bot$. (i) is proved by $\bot^+$, (ii) and (iii) are trivial and (iv) does not apply.

Case $\mathrm{at}(r)$. In this case $D^F = D$ and $G^F = G$, hence (i), (ii) and (iv) are trivial and (iii) does not apply.

For the inductive steps, assume that by induction hypothesis we have proofs M_* with $*$ among (i)–(iv).

Case $D \to G$. Only (ii) and (iv) apply. For (iv) we can use the induction hypothesis, because $D \to G \in \mathcal{I}$ exactly when $G \in \mathcal{I}$:

$$\mathcal{P}_{(iv)} := \lambda u^{D \to G}\, \lambda v^{D^F}\, M_{(iv)}(u(M_{(i)}v)).$$

In order to prove (ii), we consider subcases on clause 2.

Subcase $D \in \mathrm{MA}_0^\omega$. By Corollary 1.55, we can use case distinction on $D^F \in \mathrm{NA}_0^\omega$ to prove (ii). We thus define:

$$\begin{aligned}
\mathcal{P}_{(ii)} &:= \lambda u^{D \to G}\, \lambda v^{(D^F \to G^F) \to \bot}\, \mathsf{CD}_{D^F,\bot}\, (\lambda w^{D^F}\, \mathcal{P}^+_{(ii)}(M_{(i)}w))\, \mathcal{P}^-_{(ii)}, \text{ where}\\
\mathcal{P}^+_{(ii)} &:= \lambda w^D\, M_{(ii)}\,(uw)\,(\lambda z^{G^F}\, v(\lambda a^{D^F}\, z)),\\
\mathcal{P}^-_{(ii)} &:= \lambda w^{D^F \to \mathrm{F}}\, v(\lambda z^{D^F}\, \mathsf{efq}_{G^F}\,(wz)).
\end{aligned}$$

Subcase $D \in \mathcal{R}$. We can use the induction hypothesis for (iii). We thus define

$$\mathcal{P}_{(ii)} := \lambda u^{D \to G}\, \lambda v^{(D^F \to G^F) \to \bot}\, \mathcal{P}^+_{(ii)}(M_{(iii)}\mathcal{P}^-_{(ii)}), \text{ with } \mathcal{P}^\pm_{(ii)} \text{ defined as above.}$$

Subcase $G \in \mathcal{I}$. Then $D \to G \in \mathcal{I}$ and we can reuse the proof for (iv). We thus define

$$\mathcal{P}_{(ii)} := \lambda u^{D \to G}\, \lambda v^{(D^F \to G^F) \to \bot}\, v(\mathcal{P}_{(iv)}u).$$

Case $G \to D$. Only (i) and (iii) apply. For (iii) we can use the induction hypothesis, because $G \to D \in \mathcal{R}$ exactly when $D \in \mathcal{R}$. We define

$$\mathcal{P}_{(iii)} := \lambda u^{((G^F \to D^F) \to \mathrm{F}) \to \bot}\, \lambda v^G\, M_{(iii)}(\lambda w^{D^F \to \mathrm{F}}\, M_{(ii)} v(\lambda z^{G^F}\, u(\lambda a^{G^F \to D^F}\, w(az))))$$

In order to prove (i), we consider subcases on clause 3.

Subcase $G \in \mathcal{I}$. We can use the induction hypothesis for (iv). We thus define

$$\mathcal{P}_{(i)} := \lambda u^{G^F \to D^F}\, \lambda v^G\, M_{(i)}(u(M_{(iv)} v)).$$

Subcase $D \in \mathcal{R}$. We can use the induction hypothesis for (iii). We thus define

$$\mathcal{P}_{(i)} := \lambda u^{G^F \to D^F}\, \mathcal{P}_{(iii)}(\lambda v^{(G^F \to D^F) \to \mathrm{F}}\, \bot^+(vu)).$$

Case $\forall x\, G$ for $G \in \mathcal{I}$. Only (ii) and (iv) apply. We can use the induction hypothesis for (iv) and define:

$$\begin{aligned} \mathcal{P}_{(iv)} &:= \lambda u^{\forall x\, G}\, \lambda x\, M_{(iv)}, \\ \mathcal{P}_{(ii)} &:= \lambda u^{\forall x\, G}\, \lambda v^{\forall x\, G^F \to \bot}\, v(\mathcal{P}_{(iv)} u). \end{aligned}$$

Case $\forall x\, D$. Only (i) and (iii) apply. We can use the induction hypothesis for (iii), because $\forall x\, D \in \mathcal{R}$ exactly when $D \in \mathcal{R}$. We thus define:

$$\begin{aligned} \mathcal{P}_{(i)} &:= \lambda u^{\forall x\, D}\, \lambda x\, M_{(i)}(ux), \\ \mathcal{P}_{(iii)} &:= \lambda u^{((\forall x\, D^F) \to \mathrm{F}) \to \bot}\, \lambda x\, M_{(iii)}(\lambda v^{D^F \to \mathrm{F}}\, u(\lambda w^{\forall x\, D^F}\, v(wx))). \end{aligned}$$ □

Theorem 2.13 (Refined A-translation [BBS02]). *Let $D \in \mathcal{D}$ and $G \in \mathcal{G}$ be such that $\vdash D \to \tilde{\exists} y\, G$. Then $\vdash D^F \to \exists y\, G^F$.*

Proof. Let M' be a proof of $D \to \tilde{\exists} y\, G$. Then, using Lemma 2.12, we can find a proof of $D^F \to \forall y\, (G^F \to \bot) \to \bot$:

$$M := \lambda u^{D^F}\, \lambda v^{\forall y\, (G^F \to \bot)}\, M'(\mathcal{P}_{(i)} u)(\lambda y\, \lambda w^G\, \mathcal{P}_{(ii)}\, w\, (vy)).$$

Finally, $\lambda u^{D^F}\, M \left[\bot := \exists y\, G^F\right] u\, \exists^+_{y, G^F}$ proves $D^F \to \exists y\, G^F$. □

Although the Theorem is already general enough, in practice we will often use an equivalent but more convenient formulation, which is in fact the original statement of the theorem.

Corollary 2.14 ([BBS02]). *Let $\vec{D} \in \mathcal{D}$ and $\vec{G} \in \mathcal{G}$ be such that $\vdash \vec{D} \to \tilde{\exists} \vec{y}\, \vec{G}$. Then $\vdash \vec{D}^{\mathrm{F}} \to \exists \vec{y}\, \vec{G}^{\mathrm{F}}$.*

Proof. Similarly to above, we find a proof of $\vec{D}^{\mathrm{F}} \to \forall \vec{y}\,(\vec{G}^{\mathrm{F}} \to \bot) \to \bot$:

$$M := \lambda \vec{u}^{\vec{D}^{\mathrm{F}}}\, \lambda v^{\forall \vec{y}\,(\vec{G}^{\mathrm{F}} \to \bot)}\, M'(\overrightarrow{\mathcal{P}_{(i)} u_i})(\lambda \vec{y}\, \lambda \vec{w}^{\vec{G}}\, N_1), \text{ where}$$

$$N_i^{\bot} := \begin{cases} v\vec{y}\vec{z}, & \text{if } i > \left|\vec{G}\right|, \\ \mathcal{P}_{(ii)}^{G_i \to (G_i^F \to \bot) \to \bot} w_i(\lambda z_i^{G_i^F}\, N_{i+1}), & \text{otherwise.} \end{cases}$$

Then we define a proof $\mathcal{P}$ of $\vec{D}^{\mathrm{F}} \to \exists \vec{y}\, \vec{G}^{\mathrm{F}}$ as follows:

$$\begin{aligned} \mathcal{P} &:= \lambda \vec{u}^{\vec{D}^{\mathrm{F}}}\, M\left[\bot := \exists \vec{y}\, \vec{G}^{\mathrm{F}}\right] \vec{u}\,(\lambda \vec{y}\, \lambda \vec{v}^{\vec{G}^{\mathrm{F}}}\, \mathcal{P}_{\vec{y}}), \text{ where} \\ \mathcal{P}_x &:= \exists^+_{x, \vec{G}^{\mathrm{F}}}\, x\, \langle \vec{v} \rangle\,, \\ \mathcal{P}_{x,\vec{y}} &:= \exists^+_{x, \exists \vec{y}\, \vec{G}^{\mathrm{F}}}\, x\, \mathcal{P}_{\vec{y}}. \end{aligned}$$

□

Modified realisability and the translation from Theorem 2.13 are implemented in the proof assistant MINLOG. In Chapter 3 we will analyse some case studies for extraction via refined A-translation.

Let us return to the original problem, stated earlier in the section: given a proof of $C := \vec{D} \to \tilde{\exists} y\, G$ in NA^{ω}, we would like to find a witness t for y. Theorem 2.13 guarantees that if we search among the definite formulas $\vec{D}'$ and goal formulas G', for which $(*)$ $D_i \leftrightarrow (D_i')^F$ and $G \leftrightarrow (G')^F$, and we are able to carry out a proof M' of $C' := \vec{D}' \to \tilde{\exists} y\, G'$ in MA^{ω}, then we will automatically obtain a witness for y. One heuristic technique for finding such a proof is the following. Start from $D_i' := D_i$ and $G' := G$ and attempt to prove C'. This will not always be successful: for example, the proof can get "stuck" on a goal $\bot \to \mathrm{at}(r)$, which we cannot prove in minimal logic. In case this occurs, we can trace the occurrence of $\mathrm{at}(r)$ back to one of D_i' or G'. If r is ff, we can convert this occurrence of F to $\bot$ while aiming to continue the proof. In case r is not syntactically equal to ff, we can selectively apply the $(\cdot)^{\neg\neg}$ translation only to the troublesome $\mathrm{at}(r)$. Then $\bot \to \neg\neg\mathrm{at}(r)$ will be trivially provable and $(*)$ will be preserved, since Stability holds for NA^{ω}. In this way we effectively obtain a partial application of $(\cdot)^{\neg\neg}$, which is fine tuned for the specific case, so that $\bot$ is introduced only when it is needed in the proof. Naturally, the described steps do not constitute a formal algorithm; instead, they rely on the experience and skill of the person proving the desired formula.

It turns out that programs extracted via (refined) A-translation adhere to the *continuation passing style* [DF92]. This was first noticed by Murthy [Mur91] and later discussed also in [Mak06, Rat10]. The underlying reason is that a classical proof of existence corresponds to a minimal logic proof of "weak existence" $\tilde{\exists} x\, A$, which derives a contradiction from the assumption $\forall x\,(A \to \bot)$, by regarding $\bot$ an abstract predicate symbol. The corresponding program will have the type

$$\begin{tikzcd}
\mathrm{HA}^\omega + \mathsf{stab} \vdash \exists x^\rho\, A \arrow[r, "\text{Griffin's realisability}"] \arrow[d, "\text{negative translation}"] & \lambda\text{-}\mathcal{C}\text{-term} \arrow[d, "\text{CPS translation}"] \\
\mathrm{MA}^\omega \vdash \tilde{\exists} x^\rho\, A \arrow[r, "\text{Kreisel's realisability}"] \arrow[d, "A\text{-translation}"] & \text{CPS style } \lambda\text{-term} \arrow[d, "\text{Murthy's continuation}"] \\
\mathrm{HA}^\omega \vdash \exists x^\rho\, A \arrow[r, "\text{Kreisel's realisability}"] & \lambda\text{-term of type } \rho
\end{tikzcd}$$

Figure 2.1: Relations between A-translation and CPS

$$\tau^\circ(\tilde{\exists} x^\rho\, A) = \tau^\circ(\forall x^\rho\, (A \to \bot) \to \bot) = (\rho \Rightarrow \alpha_\bot) \Rightarrow \alpha_\bot,$$

where $\alpha_\bot$ is to be substituted with the type of the final witness to be computed. The parameter $\rho \Rightarrow \alpha_\bot$ can be viewed as a *continuation*, i.e., a function determining how to proceed in case we are given a value of type ρ. The advantage of continuations is that they can be used several times, thus effectively memorising a certain state of the program so that it can be restored later at any time, in the way this is done by Felleisen's *control operators* [FF89]. On the other hand, Griffin [Gri90] showed that the control operators can be used to extend the Curry-Howard correspondence to classical proofs by interpreting the stability axiom as a control operator. Finally, Murthy closed the loop by explicitly defining the continuation, to which a program in CPS style needs to be applied, so that the outcome is exactly a witness for $\exists x\, A$. The described correspondences are summarised in Figure 2.1.

2.3 Gödel's "Dialectica" interpretation

Gödel's functional interpretation [Göd58] occupied his interest and research for some thirty years since its first presentation in a 1941 lecture. During this time he continued reformulating and improving it, reportedly being never completely satisfied with the result [AF98]. Gödel's main motivation was to prove consistency of HA using only finitary means. As he perceived Heyting's notion of an intuitionistic proof as too abstract, he reduced the logic to a quantifier-free fragment enriched with functionals of finite type. Gödel defines the D-translation of a first-order formula $A(\vec{z})$ to a Σ_2^0 formula $A^D(\vec{z}) := \exists \vec{x} \forall \vec{y} A_D(\vec{x}, \vec{y}, \vec{z})$, where A_D is quantifier-free and $\vec{x}, \vec{y}, \vec{z}$ are tuples of variables ranging over functionals of finite type. Then the following become equivalent:

1. $A(\vec{z})$ is provable in HA
2. $A^D(\vec{z})$ is provable in HA^ω
3. There is a tuple of terms $\vec{t}$, such that $A_D(\vec{t}, \vec{y}, \vec{z})$ is provable in HA_0^ω.

A feature of the interpretation, already noted in [Göd58], is its ability to interpret Peano Arithmetic when combined with Gödel's negative translation. Thus every proof in classical arithmetic can be associated with a functional of finite type, which can be referred to as the *computational content* of the proof. One justification for such viewpoint is the case of Π_2^0 formulas, for which Gödel's negative translation and the D-translation commute. Consequently, Π_0^2 formulas are equiderivable in Peano and Heyting arithmetic and the functionals obtained from the proof are of type degree 1, i.e., computable functions over base types providing witnesses for the existential quantifiers.

Gödel's original interpretation was analysed and extended by many authors, including (but not limited to) Kreisel, Schoenfield, Howard, Diller and Nahm, Troelstra, Avigad, Feferman, Kohlenbach, Ferreira, Schwichtenberg. In our presentation we will refer primarily to [Tro73], where the interpretation is extended to HA^ω. An interpretation of NA^ω is automatically obtained by restricting the language of formulas. A main presentational difference from [Tro73] is in the use of natural deduction proof system as opposed to a Hilbert-style system, in which the Dialectica interpretation is usually formulated. Earlier natural deduction formulations of the interpretation were studied by Jørgensen [Jø01] and Hernest [Her07b]. We will follow closely the presentation in [Sch08].

2.3.1 Definition

We translate each formula $A \in \mathrm{HA}^\omega$ to a quantifier-free formula $|A|_y^x \in \mathrm{NA}_0^\omega$, connecting a realising variable $x : \tau^+(A)$ and a challenging variable $y : \tau^-(A)$. We refer to the types $\tau^+(A)$ and $\tau^-(A)$ as positive and negative computational types of A. The Dialectica interpretation starts from a proof M and produces a witnessing term t, not containing the challenging variable y freely, together with a verifying proof of $\forall y \; |A|_y^t$.

Definition 2.15 (Dialectica computational types)**.** Let $A \in \mathrm{HA}^\omega$. We define the *positive and negative computational types* of A as follows:

A	$\tau^+(A)$	$\tau^-(A)$
$\mathrm{at}(t)$	I	I
$B \to C$	$(\tau^+(B) \Rightarrow \tau^+(C)) \times (\tau^+(B) \Rightarrow \tau^-(C) \Rightarrow \tau^-(B))$	$\tau^+(B) \times \tau^-(C)$
$B \wedge C$	$\tau^+(B) \times \tau^+(C)$	$\tau^-(B) \times \tau^-(C)$
$\forall x^\rho B$	$\rho \Rightarrow \tau^+(B)$	$\rho \times \tau^-(B)$
$\exists x^\rho B$	$\rho \times \tau^+(B)$	$\tau^-(B)$

We will use the following vocabulary:

A	a witness	a challenge
requires	if $\tau^+(A) \neq \mathsf{I}$	if $\tau^-(A) \neq \mathsf{I}$
does not require	if $\tau^+(A) = \mathsf{I}$	if $\tau^-(A) = \mathsf{I}$

Proposition 2.16. *Let A be a formula in* HA^ω*. Then*

$$\begin{array}{lll}
\textit{A requires a } \dfrac{\textit{witness}}{\textit{challenge}} & \textit{iff it has} & \textit{a } \dfrac{\textit{positive}}{\textit{negative}} \textit{ existential subformula or} \\[2ex]
 & & \textit{a } \dfrac{\textit{negative}}{\textit{positive}} \textit{ universal subformula.}
\end{array}$$

Proof. We prove the claim by simultaneous induction on the formula A.

Case $\mathrm{at}(t)$. A requires neither challenges nor witnesses, so the claim is trivial.

Case $B \to C$. By the definition of $\tau^\pm(A)$ we see that

$$\begin{array}{lll}
A \text{ requires a } \dfrac{\text{witness}}{\text{challenge}} & \text{iff} & C \text{ requires a } \dfrac{\text{witness}}{\text{challenge}} \text{ or } B \text{ requires a } \dfrac{\text{challenge}}{\text{witness}} \\[2ex]
\text{(by IH)} & \text{iff} & C \text{ has a } \dfrac{\text{positive}}{\text{negative}} \text{ or} \\[2ex]
 & & B \text{ has a } \dfrac{\text{negative}}{\text{positive}} \text{ existential subformula or} \\[2ex]
 & & C \text{ has a } \dfrac{\text{negative}}{\text{positive}} \text{ or} \\[2ex]
 & & B \text{ has a } \dfrac{\text{positive}}{\text{negative}} \text{ universal subformula} \\[2ex]
\text{(by definition)} & \text{iff} & A \text{ has a } \dfrac{\text{positive}}{\text{negative}} \text{ existential subformula} \\[2ex]
 & & A \text{ has a } \dfrac{\text{negative}}{\text{positive}} \text{ universal subformula.}
\end{array}$$

Case $B \wedge C$. By the definition of $\tau^\pm(A)$ we see that

$$\begin{array}{lll}
A \text{ requires a } \dfrac{\text{witness}}{\text{challenge}} & \text{iff} & B \text{ requires a } \dfrac{\text{witness}}{\text{challenge}} \text{ or } C \text{ requires a } \dfrac{\text{witness}}{\text{challenge}} \\
\text{(by IH)} & \text{iff} & B \text{ or } C \text{ has a } \dfrac{\text{positive}}{\text{negative}} \text{ existential subformula or} \\
 & & B \text{ or } C \text{ has a } \dfrac{\text{negative}}{\text{positive}} \text{ universal subformula} \\
\text{(by definition)} & \text{iff} & A \text{ has a } \dfrac{\text{positive}}{\text{negative}} \text{ existential subformula or} \\
 & & A \text{ has a } \dfrac{\text{negative}}{\text{positive}} \text{ universal subformula.}
\end{array}$$

Case $\forall x\, B$. By definition A is universal and always requires a challenge. On the other hand A requires a witness iff B requires a challenge and since positive existential subformulas and negative universal subformulas in A and B coincide, we have proved the claim.

Case $\exists x\, B$. By definition A is existential and always requires a witness. On the other hand A requires a challenge iff B requires a challenge and since positive universal subformulas and negative existential subformulas in A and B coincide, we have proved the claim. □

Definition 2.17 (Dialectica translation)**.** Let $A \in \mathrm{HA}^\omega$ and let $r : \tau^+(A)$ and $s : \tau^-(A)$. We define the *Dialectica translation* $|A|^r_s$ as follows:

$$\begin{aligned}
|\mathrm{at}(t)|^\varepsilon_\varepsilon &:= \mathrm{at}(t), \\
|B \to C|^r_s &:= |B|^{s_\llcorner}_{r_\lrcorner(s_\llcorner)(s_\lrcorner)} \to |C|^{r_\llcorner(s_\llcorner)}_{s_\lrcorner}, \\
|B \wedge C|^r_s &:= |B|^{r_\llcorner}_{s_\llcorner} \mathbin{\tilde{\wedge}} |C|^{r_\lrcorner}_{s_\lrcorner}, \\
|\forall x^\rho\, B|^r_s &:= |B\,[x := s_\llcorner]|^{r(s_\llcorner)}_{s_\lrcorner}, \\
|\exists x^\rho\, B|^r_s &:= |B\,[x := r_\llcorner]|^{r_\lrcorner}_{s}.
\end{aligned}$$

We call r a *witness* for A and s a *challenge* for A.

The following proposition is easily proved by induction on A.

Proposition 2.18. *For every formula A in* HA^ω *and terms r, s:*

1. $\tau^\pm(A\,[x := r]) = \tau^\pm(A)$,
2. $\mathsf{FV}(|A|^r_s) \subseteq \mathsf{FV}(A) \cup \mathsf{FV}(r) \cup \mathsf{FV}(s)$
3. $|A|^r_s$ *is a formula in* NA^ω_0.
4. $|A|^\varepsilon_\varepsilon = A$ *for every formula A in* NA^ω_0.

2.3.2 Soundness

Oliva suggested an intuition for Dialectica as a game of two players: Eloise, playing the positive side ($\exists$) and Abelard, playing the negative side ($\forall$) [Oli08]. The formula A can be viewed as the game being played, and $\tau^+(A)$ and $\tau^-(A)$ specify the valid moves. Eloise plays the first move: a realiser x, which is challenged by Abelard's move y. The decidable translation $|A|^x_y$ determines whether Eloise wins or not by looking at the outcome of the games defined by the subformulas of A and submoves obtained by combining x and y. Therefore, even if A seems to be only a one move game, actually the players have to think as many moves ahead, as the formula depth is.

Now assume that A has a proof M. Then the soundness theorem for the interpretation provides Eloise with a winning strategy — a move t which beats all possible moves of Abelard. The underlying idea is that the proof M is a recipe for winning a game A by looking only at a finite number of subgames being played. Since determining the winner of every such game can be computed, Eloise can prepare for all possible subgame moves of Abelard in advance, even though he has the advantage of seeing her move by playing second.

Lemma 2.19 (Dialectica case distinction)**.** Let C be a formula, let $x : \tau^+(C)$ be a variable and let $t_1, t_2 : \tau^-(C)$ be terms. Then there is a term t such that $\vdash |C|^x_t \to |C|^x_{t_i}$ for $i = 1, 2$.

Proof. We define a *counterexample combinator*, whose intuitive purpose is to select a counterexample for a specific formula among two terms by case distinction on the decidable Dialectica translation of the formula C. We define[2]:

$$t_1 \overset{C,x}{\bowtie} t_2 := \begin{cases} t_1, & \text{if } t_1 \equiv t_2, \\ \mathsf{Cases}(|C|^x_{t_1})^{\mathrm{at}} t_2 t_1, & \text{otherwise,} \end{cases}$$

where $(\cdot)^{\mathrm{at}}$ denotes the atomic translation from Definition 1.53.

Let $D_i := |C|^x_{t_i}$. We need to construct proofs terms $\mathcal{Q}_i : |C|^x_{t_1 \overset{C,x}{\bowtie} t_2} \to D_i$ for $i = 1, 2$. We use case distinction on the decidable Dialectica translation D_1 via Theorem 1.48. Also, by Proposition 1.54 we have proof terms $K : \mathrm{at}(D_1^{\mathrm{at}}) \to D_1$ and $L : D_1 \to \mathrm{at}(D_1^{\mathrm{at}})$. Hence, we can define:

$$\begin{aligned}
\mathcal{Q}_i &:= \lambda u^{D_i} u, \text{ for } i = 1, 2, \text{ if } t_1 \equiv t_2, \text{ or otherwise,} \\
\mathcal{Q}_1 &:= \mathsf{CD}_{b,F_1} D_1^{\mathrm{at}} (\lambda u^{\mathrm{at}(D_1^{\mathrm{at}})} \lambda v^{D_2} K u)(\lambda u^{\mathrm{at}(D_1^{\mathrm{at}}) \to \mathrm{F}} \lambda w^{D_1} w), \\
\mathcal{Q}_2 &:= \mathsf{CD}_{b,F_2} D_1^{\mathrm{at}} (\lambda u^{\mathrm{at}(D_1^{\mathrm{at}})} \lambda v^{D_2} v)(\lambda u^{\mathrm{at}(D_1^{\mathrm{at}}) \to \mathrm{F}} \lambda w^{D_1} \mathsf{efq}_{D_2}(u(Lw))),
\end{aligned}$$

[2]The special case of $t_1 \equiv t_2$ is defined separately only for efficiency reasons, i.e., avoiding the case distinction when it is obviously redundant.

where $F_i := |C|^x_{\mathsf{Cases}\, b\, t_2\, t_1} \to D_i$. □

Remark 2.20. Whenever x is clear from the context, we will write $\overset{C}{\bowtie}$ instead of $\overset{C,x}{\bowtie}$.

Theorem 2.21 (Soundness of the Dialectica interpretation). *Let $A \in \mathrm{HA}^\omega$ be a formula and let $\mathcal{P}^A$ be a proof term with assumptions among $\{u_i : C_i\}_{i \geq 1}$. Let us have fresh witnessing variables $X = \{x_i : \tau^+(C_i)\}$, each one associated uniquely with an assumption variable u_i and let $y_A : \tau^-(A)$ be a fresh challenging variable associated uniquely with the formula A. Then there are terms $[\![\mathcal{P}]\!]^-_i : \tau^-(C_i)$ and $[\![\mathcal{P}]\!]^+ : \tau^+(A)$ and a proof $\overline{\mathcal{P}} : |A|^{[\![\mathcal{P}]\!]^+}_{y_A}$, such that*

1. $\mathsf{FA}(\overline{\mathcal{P}}) \subseteq \left\{ v_i : |C_i|^{x_i}_{[\![\mathcal{P}]\!]^-_i} \right\}$, *where each v_i is associated with the corresponding u_i,*
2. $\mathsf{FV}([\![\mathcal{P}]\!]^-_i), \mathsf{FV}(\overline{\mathcal{P}}) \subseteq \mathsf{FV}(\mathcal{P}) \cup X \cup \{y_A\}$,
3. $\mathsf{FV}([\![\mathcal{P}]\!]^+) \subseteq \mathsf{FV}(\mathcal{P}) \cup \{x_i\}$.

Proof. The proof proceeds by induction on the proof term $\mathcal{P}$.

Case u_1^A. Set $[\![\mathcal{P}]\!]^-_1 := y_A$, $[\![\mathcal{P}]\!]^+ := x_1$ and $\overline{\mathcal{P}} := v_1$. The variable conditions are obviously satisfied.

Case $\lambda u_0^B\, M^C$. By induction hypothesis we have a proof term $\overline{M} : |C|^{[\![M]\!]^+}_{y_C}$ with assumptions $w_0 : |B|^{x_0}_{[\![M]\!]^-_0}$ and $w_i : |C_i|^{x_i}_{[\![M]\!]^-_i}$ for $i \geq 1$. Since by definition

$$|A|^{[\![\mathcal{P}]\!]^+}_{y_A} = |B|^{y_A\llcorner}_{[\![\mathcal{P}]\!]^+\lrcorner(y_A\llcorner)(y_A\lrcorner)} \to |C|^{[\![\mathcal{P}]\!]^+\llcorner(y_A\llcorner)}_{y_A\lrcorner},$$

we will use the substitution $\xi := [x_0 := y_A\llcorner]\,[y_C := y_A\lrcorner]$ and define

$$\begin{aligned} [\![\mathcal{P}]\!]^+ &:= \left\langle \lambda x_0\, [\![M]\!]^+, \lambda x_0\, \lambda y_C\, [\![M]\!]^-_0 \right\rangle, \\ [\![\mathcal{P}]\!]^-_i &:= [\![M]\!]^-_i \xi, \qquad \text{for } i \geq 1, \\ \overline{\mathcal{P}} \quad &:= (\lambda w_0\, \overline{M})\xi, \ \text{with } v_i := w_i\xi. \end{aligned}$$

The variable conditions are satisfied, because $\mathsf{FV}([\![\mathcal{P}]\!]^-_i) = \mathsf{FV}([\![M]\!]^-_i) \cup \{y_A\} \setminus \{x_0, y_C\}$ and $\mathsf{FV}([\![\mathcal{P}]\!]^+) = \mathsf{FV}([\![M]\!]^+) \cup \mathsf{FV}([\![M]\!]^-_0) \setminus \{x_0, y_C\}$.

Case $M_1^{C \to A} M_2^C$. Let us denote $B := C \to A$. By induction hypothesis we have proof terms

$$\begin{aligned} &\overline{M}_1 : |C|^{[\![M_1]\!]^+}_{y_B} \text{ with assumptions } w_i' : |C_i|^{x_i}_{[\![M_1]\!]^-_i} \text{ and} \\ &\overline{M}_2 : |B|^{[\![M_2]\!]^+}_{y_C} \text{ with assumptions } w_i'' : |C_i|^{x_i}_{[\![M_2]\!]^-_i}. \end{aligned}$$

Note that by definition

$$|B|^{[\![M_1]\!]^+}_{y_B} = |C|^{y_B\llcorner}_{[\![M_1]\!]^+\lrcorner(y_B\llcorner)(y_B\lrcorner)} \to |A|^{[\![M_1]\!]^+\llcorner(y_B\llcorner)}_{y_B\lrcorner}.$$

We will thus use the substitutions

$$\begin{aligned} \xi_1 &:= \big[y_B := \big\langle \llbracket M_2 \rrbracket^+, y_A \big\rangle\big] && \text{for } M_1, \\ \xi_2 &:= \big[y_C := \llbracket M_1 \rrbracket^+ {\lrcorner} \llbracket M_2 \rrbracket^+ y_A\big] && \text{for } M_2. \end{aligned}$$

However, for every shared assumption variable $u_i \in \mathsf{FA}(M_1) \cap \mathsf{FA}(M_2)$ we have two candidates for counterexamples: $\llbracket M_j \rrbracket_i^- \xi_j$ for $j = 1, 2$. We need to construct $\llbracket \mathcal{P} \rrbracket_i^-$ such that $|C_i|^{x_i}_{\llbracket \mathcal{P} \rrbracket_i^-}$ implies both $|C_i|^{x_i}_{\llbracket M_j \rrbracket_i^- \xi_j}$. We apply Lemma 2.19 and define:

$$\begin{aligned} \llbracket \mathcal{P} \rrbracket^+ &:= \llbracket M_1 \rrbracket^+ {\llcorner} \llbracket M_2 \rrbracket^+, \\ \llbracket \mathcal{P} \rrbracket_i^- &:= \llbracket M_1 \rrbracket_i^- \xi_1 \overset{C_i}{\bowtie} \llbracket M_2 \rrbracket_i^- \xi_2. \end{aligned}$$

In order to unify treatment of all assumption variables, we assume that $\llbracket M_j \rrbracket_i^- := \llbracket M_{3-j} \rrbracket_i^-$ whenever $u_i \in \mathsf{FA}(M_{3-j}) \setminus \mathsf{FA}(M_j)$. By Lemma 2.19 we have proof terms $\mathcal{Q}_i^{(j)} : |C_i|^{x_i}_{\llbracket \mathcal{P} \rrbracket_i^-} \to |C_i|^{x_i}_{\llbracket M_j \rrbracket_i^- \xi_j}$ and using them we define:

$$\overline{\mathcal{P}} := \overline{\mathcal{P}}_1 \overline{\mathcal{P}}_2, \qquad \text{where } \overline{\mathcal{P}}_j := \overline{M}_j \xi_j \vec{\eta}_j \text{ with } \eta_{j,i} := \Big[w_i^{(j)} := \mathcal{Q}_i^{(j)} v_i\Big].$$

The variable conditions are satisfied, because $\mathsf{FV}(\llbracket \mathcal{P} \rrbracket^+) \subseteq \mathsf{FV}(\llbracket M_1 \rrbracket^+) \cup \mathsf{FV}(\llbracket M_2 \rrbracket^+)$ and $\mathsf{FV}(\llbracket \mathcal{P} \rrbracket_i^-) \subseteq \{y_A\} \cup (\mathsf{FV}(\llbracket M_1 \rrbracket_i^-) \setminus \{y_B\}) \cup (\mathsf{FV}(\llbracket M_2 \rrbracket_i^-) \setminus \{y_C\})$.

Case $\lambda x^\rho\, M^B$. By induction hypothesis we have a proof of $\overline{M} : |B|^{\llbracket M \rrbracket^+}_{y_B}$ with assumptions $w_i : |C_i|^{x_i}_{\llbracket M \rrbracket_i^-}$. Since by definition

$$|A|^{\llbracket \mathcal{P} \rrbracket^+}_{y_A} = |B\,[x := y_A{\llcorner}]|^{\llbracket \mathcal{P} \rrbracket^+ (y_A{\llcorner})}_{y_A{\lrcorner}},$$

we can substitute $\xi := [x := y_A{\llcorner}]\,[y_B := y_A{\lrcorner}]$. Thus we define $\llbracket \mathcal{P} \rrbracket^+ := \lambda x\, \llbracket M \rrbracket^+$, $\llbracket \mathcal{P} \rrbracket_i^- := \llbracket M \rrbracket_i^- \xi$ and $\overline{\mathcal{P}} := \overline{M} \xi$ with $v_i := w_i \xi$. The variable conditions are satisfied, because $\mathsf{FV}(\llbracket \mathcal{P} \rrbracket^+) := \mathsf{FV}(\llbracket M \rrbracket^+) \setminus \{x\}$ and $\mathsf{FV}(\llbracket \mathcal{P} \rrbracket_i^-) := \mathsf{FV}(\llbracket M \rrbracket_i^-) \cup \{y_A\} \setminus \{x, y_B\}$.

Case $M^{\forall x\, B} t$. Let $C := \forall x\, B$. By induction hypothesis we have a proof $\overline{M}$ of

$$|C|^{\llbracket M \rrbracket^+}_{y_C} = |B\,[x := y_C{\llcorner}]|^{\llbracket M \rrbracket^+ (y_C{\llcorner})}_{y_C{\lrcorner}}$$

with assumptions $w_i : |C_i|^{x_i}_{\llbracket M \rrbracket_i^-}$. Since $A = B\,[x := t]$, we can use $\xi := [y_C := \langle t, y_A \rangle]$. We thus define $\llbracket \mathcal{P} \rrbracket^+ := \llbracket M \rrbracket^+ t$, $\llbracket \mathcal{P} \rrbracket_i^- := \llbracket M \rrbracket_i^- \xi$ and $\overline{\mathcal{P}} := \overline{M} \xi$ with $v_i := w_i \xi$. The variable conditions are satisfied, because $\mathsf{FV}(\llbracket \mathcal{P} \rrbracket^+) = \mathsf{FV}(\llbracket M \rrbracket^+) \cup \mathsf{FV}(t)$ and $\mathsf{FV}(\llbracket \mathcal{P} \rrbracket_i^-) = \mathsf{FV}(\llbracket M \rrbracket_i^-) \cup \mathsf{FV}(t) \cup \{y_A\} \setminus \{y_C\}$.

Case AxT. Define $\llbracket \mathsf{AxT} \rrbracket^+ := \varepsilon$, $\overline{\mathsf{AxT}} := \mathsf{AxT}$.

Because of the relatively complex interpretation of implication, we introduce a technical simplification for the rest of the axioms by treating the corresponding rules instead. We will illustrate with an example that we do not lose generality in this way. Suppose that we have an instance of the induction rule $\mathcal{P} := \mathsf{Ind}_{\mathsf{N}}^{n,A}\, n\, M_1^{A[n:=0]}\, (\lambda n\, \lambda u_0^A\, M_2^{A[n:=\mathsf{S}n]})$ and $\llbracket \mathcal{P} \rrbracket^+, \llbracket \mathcal{P} \rrbracket_i^-, \overline{\mathcal{P}}$ are already defined. We consider the proof $\mathcal{Q} := \lambda n\, \lambda u_1\, \lambda u_2\, \mathcal{P}$, where $M_j := u_j$ for $j = 1, 2$ with u_j fresh assumption variables. Thus, we can postulate $\llbracket \mathsf{Ind}_{\mathsf{N}}^{n,A} \rrbracket^+ := \llbracket \mathcal{Q} \rrbracket^+$ and then $\overline{\mathsf{Ind}_{\mathsf{N}}^{n,A}} := \overline{\mathcal{Q}}$.

Case $\mathcal{C}^{b,A}\, b M_{\mathsf{tt}}^{A[b:=\mathsf{tt}]} M_{\mathsf{ff}}^{A[b:=\mathsf{ff}]}$. By induction hypothesis we have proofs

$$\overline{M}_j : |A\,[b := j]|_{y_A}^{\llbracket M_j \rrbracket^+} \text{ with assumptions } w_i^j : |C_i|_{\llbracket M_j \rrbracket_i^-}^{x_i} \text{ for } j = \mathsf{ff}, \mathsf{tt}.$$

Let us define $\llbracket \mathcal{P} \rrbracket^+ := \mathsf{Cases}\, b\, \llbracket M_{\mathsf{tt}} \rrbracket^+ \llbracket M_{\mathsf{ff}} \rrbracket^+$ and $\llbracket \mathcal{P} \rrbracket_i^- := \mathsf{Cases}\, b\, \llbracket M_{\mathsf{tt}} \rrbracket_i^-\, \llbracket M_{\mathsf{ff}} \rrbracket_i^-$. Then we can define $\overline{\mathcal{P}} := \mathcal{C}\, b\, (\lambda \vec{w}'\, \overline{M}_{\mathsf{tt}})\, (\lambda \vec{w}''\, \overline{M}_{\mathsf{ff}}) \vec{v}$, because $v_i\, [b := j]$ and w_i^j have equal formulas. The variable conditions are satisfied, because $\mathsf{FV}(\llbracket \mathcal{P} \rrbracket^+) = \mathsf{FV}(\llbracket M \rrbracket^+) \cup \mathsf{FV}(\llbracket N \rrbracket^+) \cup \{b\}$ and $\mathsf{FV}(\llbracket \mathcal{P} \rrbracket_i^-) = \mathsf{FV}(\llbracket M \rrbracket_i^-) \cup \mathsf{FV}(\llbracket N \rrbracket_i^-) \cup \{b\}$.

Case $\mathsf{Ind}_{\mathsf{N}}^{n,A}\, n\, M_1^{A[n:=0]}\, (\lambda n\, \lambda u_0^A\, M_2^{A[n:=\mathsf{S}n]})$. By induction hypothesis we have proofs

$$\begin{aligned} &\overline{M}_1 : |A\,[n := 0]|_{y_A}^{\llbracket M_1 \rrbracket^+} && \text{with assumptions } w_i' : |C_i|_{\llbracket M_1 \rrbracket_i^-}^{x_i} \text{ for } i \geq 1 \text{ and} \\ &\overline{M}_2 : |A\,[n := \mathsf{S}n]|_{y_A}^{\llbracket M_2 \rrbracket^+} && \text{with assumptions } w_0'' : |A|_{\llbracket M_2 \rrbracket_0^-}^{x_0} \text{ and } w_i'' : |C_i|_{\llbracket M_2 \rrbracket_i^-}^{x_i} \text{ for } i \geq 1. \end{aligned}$$

As before, for the sake of unified treatment let us define $\llbracket M_j \rrbracket_i^- := \llbracket M_{3-j} \rrbracket_i^-$ if $u_i \in \mathsf{FV}(M_{3-j}) \setminus \mathsf{FV}(M_j)$ for $i \geq 1$. We define

$$\begin{aligned} \llbracket \mathcal{P} \rrbracket^+ &:= \mathcal{R}_{\mathsf{N}}^{\tau^+(A)}\, n\, \llbracket M_1 \rrbracket^+ (\lambda n\, \lambda x_0\, \llbracket M_2 \rrbracket^+), \\ \llbracket \mathcal{P} \rrbracket_i^- &:= \mathcal{R}_{\mathsf{N}}^{\tau^-(A) \Rightarrow \tau^-(C_i)}\, n\, (\lambda y_A\, \llbracket M_1 \rrbracket_i^-) \Big(\lambda n\, \lambda p\, \lambda y_A\, (p \llbracket M_2 \rrbracket_0^- \overset{C_i}{\bowtie} \llbracket M_2 \rrbracket_i^-) \xi \Big) y_A, \end{aligned}$$

where $\xi := \big[x_0 := \llbracket \mathcal{P} \rrbracket^+ \big]$. We will define a proof $\mathcal{Q}$ of the formula $B := \forall y_A\, (\vec{D} \to |A|_{y_A}^{\llbracket \mathcal{P} \rrbracket^+})$, where $D_i := |C_i|_{\llbracket \mathcal{P} \rrbracket_i^-}^{x_i}$. Then we will be able to set $\overline{\mathcal{P}} := \mathcal{Q} y_A \vec{v}$.

First, we note that by definition:

$$\begin{aligned} B\,[n := 0] &= \forall y_A \left(\overrightarrow{|C_i|_{\llbracket M_1 \rrbracket_i^-}^{x_i}} \to |A\,[n := 0]|_{y_A}^{\llbracket M_1 \rrbracket^+} \right), \text{ which is proved by } \overline{M_1}, \text{ and} \\ B\,[n := \mathsf{S}n] &= \forall y_A \left(\overrightarrow{|C_i|_{t_i \xi}^{x_i}} \to |A\,[n := \mathsf{S}n]|_{y_A}^{\llbracket M_2 \rrbracket^+ \xi} \right), \end{aligned}$$

where $t_i := t_i' \overset{C_i}{\bowtie} t_i''$ with $t_i' := \llbracket \mathcal{P} \rrbracket_i^-\, \big[y_A := \llbracket M_2 \rrbracket_0^- \big]$ and $t_i'' := \llbracket M_2 \rrbracket_i^-$. We notice that $B\,[n := \mathsf{S}n]$ can be proved by $\overline{M_2} \xi$ if we are able to prove all its assumptions $w_i'' \xi$. By Lemma 2.19 we have proofs $\mathcal{Q}_i^{(j)} : |C_i|_{t_i \xi}^{x_i} \to |C_i|_{t_i^{(j)} \xi}^{x_i}$. Now $w_i'' : |C_i|_{t_i''}^{x_i}$ for $i \geq 1$, while $w_0'' \xi : |A|_{\llbracket M_2 \rrbracket_0^- \xi}^{\llbracket \mathcal{P} \rrbracket^+}$ can be obtained from B with y_A instantiated as $\llbracket M_2 \rrbracket_0^- \xi$. But then $D_i\, \big[y_A := \llbracket M_2 \rrbracket_0^- \xi \big]$ are equal exactly to $|C_i|_{t_i' \xi}^{x_i}$. We are ready to define $\mathcal{Q}$ by induction as follows:

$$\mathcal{Q} := \mathsf{Ind}_{\mathsf{N}}^{n,B}\, n\, (\lambda y_A\, \lambda \vec{w}'\, \overline{M}_1)(\lambda n\, \lambda p^B\, \lambda y_A\, \lambda \vec{v}\, (\lambda \vec{w}''\, \overline{M}_2)\xi(p[\![M_2]\!]_0^- \overrightarrow{\mathcal{Q}_i' v_i})\overrightarrow{\mathcal{Q}_i'' v_i}).$$

The variable conditions hold because $\mathsf{FV}([\![\mathcal{P}]\!]^+) \subseteq \mathsf{FV}([\![M_1]\!]^+) \cup \mathsf{FV}([\![M_2]\!]^+) \setminus \{x_0\} \cup \{n\}$ and $\mathsf{FV}([\![\mathcal{P}]\!]_i^-) \subseteq \mathsf{FV}([\![M_1]\!]_i^-) \cup \mathsf{FV}([\![M_2]\!]_i^- \setminus \{x_0\} \cup \{n\}$.

Case $\mathsf{Ind}_{\mathsf{L}(\rho)}^{l,A}\, l\, M_1^{A[l:=\mathsf{nil}]}\, (\lambda x\, \lambda l\, \lambda u_0^A\, M_2^{A[l:=x::l]})$. This case is very similar to the previous one. By induction hypothesis we have proofs

$\overline{M}_1 : |A\,[l := \mathsf{nil}]|_{y_A}^{[\![M_1]\!]^+}$ with assumptions $w_i' : |C_i|_{[\![M_1]\!]_i^-}^{x_i}$ for $i \geq 1$ and

$\overline{M}_2 : |A\,[l := x :: l]|_{y_A}^{[\![M_2]\!]^+}$ with assumptions $w_0'' : |A|_{[\![M_2]\!]_0^-}^{x_0}$ and $w_i'' : |C_i|_{[\![M_2]\!]_i^-}^{x_i}$ for $i \geq 1$.

We define

$$[\![\mathcal{P}]\!]^+ := \mathcal{R}_{\mathsf{L}(\rho)}^{\tau^+(A)}\, l\, [\![M_1]\!]^+(\lambda x\, \lambda l\, \lambda x_0\, [\![M_2]\!]^+),$$
$$[\![\mathcal{P}]\!]_i^- := \mathcal{R}_{\mathsf{L}(\rho)}^{\tau^-(A)\Rightarrow\tau^-(C_i)}\, l\, (\lambda y_A\, [\![M_1]\!]_i^-)\Big(\lambda x\, \lambda l\, \lambda p\, \lambda y_A\, (p[\![M_2]\!]_0^- \overset{C_i}{\bowtie} [\![M_2]\!]_i^-)\xi\Big) y_A,$$

where $\xi := \big[x_0 := [\![\mathcal{P}]\!]^+\big]$. We adopt the definitions of D_i, t_i, $t_i^{(j)}$ and $\mathcal{Q}_i^{(j)}$ from the previous case and, as before, define a proof $\mathcal{Q}$ of the formula $B := \forall y_A\, (\vec{D} \to |A|_{y_A}^{[\![\mathcal{P}]\!]^+})$:

$$\mathcal{Q} := \mathsf{Ind}_{\mathsf{L}(\rho)}^{l,B}\, l\, (\lambda y_A\, \lambda \vec{w}'\, \overline{M}_1)(\lambda x\, \lambda l\, \lambda p^B\, \lambda y_A\, \lambda \vec{v}\, (\lambda \vec{w}''\, \overline{M}_2)\xi(p[\![M_2]\!]_0^- \overrightarrow{\mathcal{Q}_i' v_i})\overrightarrow{\mathcal{Q}_i'' v_i}).$$

Finally, $\overline{\mathcal{P}} := \mathcal{Q} y_A \vec{v}$. The variable conditions hold as in the previous case.

Case $\langle M_1^B, M_2^C\rangle$. We adopt the notations from the case of implication elimination. Note that since by definition

$$|A|_{y_A}^{[\![\mathcal{P}]\!]^+} = |B|_{y_A\llcorner}^{[\![M_1]\!]^+} \mathbin{\tilde{\wedge}} |C|_{y_A\lrcorner}^{[\![M_2]\!]^+},$$

we will use the substitutions $\xi_1 := [y_B := y_A\llcorner]$ and $\xi_2 := [y_C := y_A\lrcorner]$. Thus we define $[\![\mathcal{P}]\!]^+ := \langle [\![M_1]\!]^+, [\![M_2]\!]^+\rangle$ and $[\![\mathcal{P}]\!]_i^- := [\![M_1]\!]_i^- \xi_1 \overset{C_i}{\bowtie} [\![M_2]\!]_i^- \xi_2$. By Lemma 2.19 we have proofs $\mathcal{Q}_i^{(j)} : |C_i|_{[\![\mathcal{P}]\!]_i^-}^{x_i} \to |C_i|_{[\![M_j]\!]_i^- \xi_j}^{x_i}$. Hence, we can define

$$\overline{\mathcal{P}} := \lambda u^{|B|_{y_A\llcorner}^{[\![M_1]\!]^+} \to |C|_{y_A\lrcorner}^{[\![M_2]\!]^+} \to \mathrm{F}}\, u\big((\lambda \vec{w}'\, \overline{M}_1)\xi_1 \overrightarrow{Q_i' v_i}\big)\, \big((\lambda \vec{w}''\, \overline{M}_2)\xi_2 \overrightarrow{Q_i'' v_i}\big).$$

The variable conditions hold, because $\mathsf{FV}([\![\mathcal{P}]\!]^+) = \mathsf{FV}([\![M_1]\!]^+) \cup \mathsf{FV}([\![M_2]\!]^+)$ and $\mathsf{FV}([\![\mathcal{P}]\!]_i^-) := \mathsf{FV}([\![M_1]\!]_i^-) \cup \mathsf{FV}([\![M_2]\!]_i^-) \setminus \{y_B, y_C\} \cup \{y_A\}$.

Case $M^{A\wedge B}\llcorner$. Set $C := A \wedge B$. By induction hypothesis we have a proof $\overline{M}$ of

$$|C|_{y_C}^{[\![M]\!]^+} = |A|_{y_C\llcorner}^{[\![M]\!]^+\llcorner} \mathbin{\tilde{\wedge}} |B|_{y_C\lrcorner}^{[\![M]\!]^+\lrcorner}$$

from assumptions $w_i : |C_i|_{[\![M]\!]_i^-}^{x_i}$. We set $\xi := [y_C := \langle y_A, y_B\rangle]$ and define

$$\llbracket\mathcal{P}\rrbracket^+ := \llbracket M\rrbracket^+\llcorner, \qquad \llbracket\mathcal{P}\rrbracket_i^- := \llbracket M\rrbracket_i^-\xi, \qquad \overline{\mathcal{P}} := \overline{M}\xi(\lambda u\,\lambda v\,u),$$

where $v_i := w_i\xi$.

Case $M^{B\wedge A}\lrcorner$. Similarly to the previous case, so we define

$$\llbracket\mathcal{P}\rrbracket^+ := \llbracket M\rrbracket^+\lrcorner, \qquad \llbracket\mathcal{P}\rrbracket_i^- := \llbracket M\rrbracket_i^-\xi, \qquad \overline{\mathcal{P}} := \overline{M}\xi(\lambda u\,\lambda v\,v),$$

where $\xi := [y_{B\wedge A} := \langle y_B, y_A\rangle]$.

Case $\left\langle t, M^{B[x:=t]}\right\rangle$. By induction hypothesis we have a proof $\overline{M} : |B\,[x := t]|_{y_B}^{\llbracket M\rrbracket^+}$ from assumptions $w_i : |C_i|_{\llbracket M\rrbracket_i^-}^{x_i}$. By definition

$$|A|_{y_A}^{\llbracket\mathcal{P}\rrbracket^+} = \left|B\left[x := \llbracket\mathcal{P}\rrbracket^+\llcorner\right]\right|_{y_A}^{\llbracket\mathcal{P}\rrbracket^+\lrcorner},$$

hence we define $\llbracket\mathcal{P}\rrbracket^+ := \left\langle t, \llbracket M\rrbracket^+\right\rangle, \llbracket\mathcal{P}\rrbracket_i^- := \llbracket M\rrbracket_i^-$ and $\overline{\mathcal{P}} := \overline{M}$ with $v_i := w_i$. The variable conditions hold since $\mathsf{FV}(\llbracket\mathcal{P}\rrbracket^+) = \mathsf{FV}(t) \cup \mathsf{FV}(\llbracket M\rrbracket^+)$ and $\mathsf{FV}(\llbracket\mathcal{P}\rrbracket_i^-) = \mathsf{FV}(\llbracket M\rrbracket_i^-)$.

Case $\exists^-_{x,C,A} M_1^{\exists x\,C}(\lambda x\,\lambda u_0^C\,M_2^A)$. We set $B := \exists x\,C$. By induction hypothesis we have proofs

$\overline{M}_1 : |B|_{y_B}^{\llbracket M_1\rrbracket^+}$ with assumptions $w_i' : |C_i|_{\llbracket M_1\rrbracket_i^-}^{x_i}$ for $i \geq 1$ and

$\overline{M}_2 : |A|_{y_A}^{\llbracket M_2\rrbracket^+}$ with assumptions $w_0'' : |C|_{\llbracket M_2\rrbracket_0^-}^{x_0}$ and $w_i'' : |C_i|_{\llbracket M_2\rrbracket_i^-}^{x_i}$ for $i \geq 1$.

By definition we have

$$|B|_{y_B}^{\llbracket M_1\rrbracket^+} = \left|C\left[x := \llbracket M_1\rrbracket^+\llcorner\right]\right|_{y_B}^{\llbracket M_1\rrbracket^+\lrcorner}.$$

We use the substitutions $\xi_2 := \left[x := \llbracket M_1\rrbracket^+\llcorner\right]\left[x_0 := \llbracket M_1\rrbracket^+\lrcorner\right]$, $\xi_1 := \left[y_B := \llbracket M_2\rrbracket_0^-\xi_2\right]$ and define

$$\begin{aligned} \llbracket\mathcal{P}\rrbracket^+ &:= \llbracket M_1\rrbracket^+\lrcorner, \\ \llbracket\mathcal{P}\rrbracket_i^- &:= \llbracket M_1\rrbracket_i^-\xi_1 \overset{C_i}{\bowtie} \llbracket M_2\rrbracket_i^-\xi_2. \end{aligned}$$

By Lemma 2.19 we have proofs $\mathcal{Q}_i^{(j)} : |C_i|_{\llbracket\mathcal{P}\rrbracket_i^-}^{x_i} \to |C_i|_{\llbracket M_j\rrbracket^-\xi_j}^{x_i}$. Hence, we can define the proof

$$\overline{\mathcal{P}} := (\lambda\vec{w}''\,\overline{M}_2)\xi_2((\lambda\vec{w}'\,\overline{M}_1)\xi_1\overrightarrow{\mathcal{Q}_i'v_i})\overrightarrow{\mathcal{Q}_i''v_i}.$$

The variable conditions are satisfied because $\mathsf{FV}(\llbracket\mathcal{P}\rrbracket_i^-) = \mathsf{FV}(\llbracket M_1\rrbracket_i^-) \cup \mathsf{FV}(\llbracket M_2\rrbracket_i^-) \setminus \{x, x_0, y_B\}$ and $\mathsf{FV}(\llbracket\mathcal{P}\rrbracket^+) = \mathsf{FV}(\llbracket M_1\rrbracket^+)$. □

Corollary 2.22 (Dialectica extraction)**.** *Let* $\vdash \forall x^\rho \,\tilde{\exists} y^\sigma \, A$*, where* $A \in \mathrm{NA}_0^\omega$*. Then there is a term* $t : \rho \Rightarrow \sigma$*, such that* $\vdash \forall x \, A\,[y := tx]$*.*

Proof. Let us denote $B := \forall x \,\tilde{\exists} y \, A$ and let $\mathcal{P}$ be a proof of B. By definition

$$\begin{aligned}
\tau^+(B) &= \rho \Rightarrow ((\sigma \Rightarrow \mathsf{I}) \Rightarrow \mathsf{I}) \times ((\sigma \Rightarrow \mathsf{I}) \Rightarrow \mathsf{I} \Rightarrow \sigma \times \mathsf{I}) = \rho \Rightarrow \sigma, \\
\tau^-(B) &= \rho \times (\sigma \Rightarrow \mathsf{I}) \times \mathsf{I} = \rho, \\
|B|_x^t &= \left| \tilde{\exists} x \, A \right|_\varepsilon^{tx} = \neg \left| \forall y \, \neg A \right|_{tx}^\varepsilon = \neg \left| \neg A\,[y := tx] \right|_{tx}^\varepsilon \\
&= \neg\neg \left| A\,[y := tx] \right|_\varepsilon^\varepsilon = \neg\neg A\,[y := tx]\,.
\end{aligned}$$

Finally, by Theorem 2.21 we have $t := [\![\mathcal{P}]\!]^+$ and a closed proof $\overline{\mathcal{P}} : A\,[y := tx]$. $\square$

CHAPTER

THREE

CASE STUDIES FOR PROGRAM EXTRACTION

In this chapter we will present a collection of case studies for program extraction from proofs in classical logic. Every case study will be analysed both with refined A-translation and the Dialectica interpretation and the resulting programs will be compared. These case studies will serve as basis and motivation for the optimised variants of the Dialectica interpretation, which will be presented in the following chapters.

Every example in this chapter starts with informal presentation of the theorem proved. Then we continue with formalising the case study and writing a proof term, modularised into suitable lemmas. Since our main goal is to compare the behaviour of refined A-translation and the Dialectica interpretation, we have to formalise the case studies in a manner, which is treatable by both techniques. We thus choose to work in MA^ω and more specifically with formulas of the shape required by refined A-translation (cf. Theorem 2.13). The Dialectica interpretation works with every HA^ω proof and in this sense is more general. We will thus translate the MA^ω proof used for refined A-translation to an NA^ω proof by substituting $[\bot := \mathrm{F}]$. Hence, we will apply Dialectica to morally the same proof, hoping to achieve an honest comparison of the two methods.

3.1 Stolzenberg's example

We will start with a simple example, which makes non-trivial use of classical logic. This case study is attributed to G. Stolzenberg and has been popularised by Coquand [Coq95]. It has become a standard simple test for methods for extraction from classical proof and has been treated by many authors (cf. [Mur91, BBS97, Urb00, Sei03, Mak06, Rat10]). We will start with an informal proof of Stolzenberg's example, which we will then formalise in MA^ω.

Lemma 3.1 (Stolzenberg)**.** Every infinite boolean sequence has an element which occurs infinitely often.

Proof. Assume that neither tt nor ff occur infinitely often. Then there are indices n_{tt} and n_{ff}, starting from which respectively tt and ff do not appear. However, this situation is impossible, because the index $\max(n_{\mathsf{tt}}, n_{\mathsf{ff}})$ is a counterexample to one of the assumptions, depending which element appears at this index. □

Corollary 3.2. *In every infinite boolean sequence there are at least two occurrences of the same element.*

Proof. By Lemma 3.1, we know that there is an element occurring infinitely often. Take its first two occurrences. □

Remark 3.3. The last corollary can be easily proved by a pigeonhole principle argument by looking at the first three elements; two of them must be equal. As noted by Coquand in [Coq95], the point of the example is to show that attempting to find a witness from this classical proof produces an asymmetric program. This is surprising, because the classical proof can be expressed completely symmetrically in an appropriate system (e.g. classical sequent calculus), but when we attempt to view the proof as expressing some calculation we are forced to make a choice, which breaks the symmetry. This choice has different expressions with the different methods: direction in which to permute the cut ([Urb00]), the use of negative translation ([Sei03, Rat10]) or order of existential elimination ([Mak06]).

3.1.1 Proof formalisation

We will formalise the proofs of Lemma 3.1 and Corollary 3.2 above in MA^ω, so that we can apply both methods on the same proof.

We will use $\mathsf{BS} := \mathsf{N} \Rightarrow \mathsf{B}$ as the type of infinite boolean sequences. We assume that we have definitions of the functions $\max^{\mathsf{N}\Rightarrow\mathsf{N}\Rightarrow\mathsf{N}}$, $<^{\mathsf{N}\Rightarrow\mathsf{N}\Rightarrow\mathsf{B}}$ and $\leq^{\mathsf{N}\Rightarrow\mathsf{N}\Rightarrow\mathsf{B}}$. For brevity we will write $n < m$ and $n \leq m$, instead of $\mathrm{at}(<nm)$ and $\mathrm{at}(\leq nm)$ and $m \sqcup n$ instead of $\max mn$.

Lemma 3.1 can be stated formally as

$$\forall f^{\mathsf{BS}}\, \tilde{\exists} b^{\mathsf{B}} \forall n^{\mathsf{N}}\, \tilde{\exists} k^{\mathsf{N}} n \leq k \mathbin{\tilde{\wedge}} fk = b.$$

A proof of the lemma can be defined as:

$$L := \lambda f\, \lambda u\; u\, \mathsf{tt}(\lambda n_{\mathsf{tt}}^{\mathsf{N}}\, \lambda v_{\mathsf{tt}}\, u\, \mathsf{ff}(\lambda n_{\mathsf{ff}}^{\mathsf{N}}\, \lambda v_{\mathsf{ff}}\, C_B(f(n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}))L_{\mathsf{tt}}L_{\mathsf{ff}}),$$

where

$$
\begin{aligned}
u &: \forall b\, \tilde{\exists} n \forall k\, (n \le k \to fk = b \to \bot), && \\
v_b &: \forall k\, (n_b \le k \to fk = b \to \bot), && \text{for } b \in \{\mathsf{tt}, \mathsf{ff}\}, \\
L_b &: f(n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}) = b \to \bot, && \text{for } b \in \{\mathsf{tt}, \mathsf{ff}\}, \\
L_b &:= v_b(n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}) L_{\max,b}, && \text{for } b \in \{\mathsf{tt}, \mathsf{ff}\}, \\
L_{\max,b} &: \forall n_{\mathsf{tt}} \forall n_{\mathsf{ff}}\, (n_b \le n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}), && \text{for } b \in \{\mathsf{tt}, \mathsf{ff}\}, \\
C_B &: \forall b\, \big((b = \mathsf{tt} \to \bot) \to (b = \mathsf{ff} \to \bot) \to \bot\big) && \\
C_B &:= \lambda b\, \mathcal{C}\, b\, M_{\mathsf{tt}}\, M_{\mathsf{ff}}, && \\
M_b &: \big((b = \mathsf{tt} \to \bot) \to (b = \mathsf{ff} \to \bot) \to \bot\big), && \text{for } b \in \{\mathsf{tt}, \mathsf{ff}\}, \\
M_b &:= \lambda w_{b,\mathsf{tt}}\, \lambda w_{b,\mathsf{ff}}\, w_{b,b} \mathsf{AxT}, && \text{for } b \in \{\mathsf{tt}, \mathsf{ff}\}, \\
w_{b_1,b_2} &: b_1 = b_2 \to \bot, && \text{for } b_1, b_2 \in \{\mathsf{tt}, \mathsf{ff}\}.
\end{aligned}
$$

Corollary 3.2 is expressed by the formula

$$\forall f^{\mathsf{BS}}\, \tilde{\exists} k_1, k_2 (k_1 < k_2 \mathbin{\tilde{\wedge}} fk_1 = fk_2).$$

The corollary is proved by

$$
\begin{aligned}
M := \lambda f\, \lambda v\, Lf(\lambda b\, \lambda w\, w0 \quad & (\lambda k_1\, \lambda u_1^{0 \le k_1} \quad \lambda z_1^{fk_1 = b} \\
& w(\mathsf{S}k_1)(\lambda k_2\, \lambda u_2^{\mathsf{S}k_1 \le k_2}\, \lambda z_2^{fk_2 = b}\, vk_1k_2(M_< v_2)(M_= z_1 z_2)))),
\end{aligned}
$$

where

$$
\begin{aligned}
v &: \forall k_1 \forall k_2\, (k_1 < k_2 \to fk_1 = fk_2 \to \bot), \\
w &: \forall n\, \tilde{\exists} k\, (n \le k \mathbin{\tilde{\wedge}} fk = b), \\
M_< &: \mathsf{S}k_1 \le k_2 \to k_1 < k_2, \\
M_= &: fk_1 = b \to fk_2 = b \to fk_1 = fk_2.
\end{aligned}
$$

Note that we have omitted the definitions of $L_{\max,b}$, $M_<$ and $M_=$, because they will be irrelevant for extraction.

We are ready to apply the extraction methods from Chapter 2. For this first and simplest example we will carry out the extraction more rigorously and for subsequent examples we will just present the final result.

3.1.2 Extraction via refined A-translation

Let $G_1 := k_1 < k_2$, $G_2 := fk_1 = fk_2$ and $\tilde{A} := \tilde{\exists} k_1, k_2\, (G_1 \mathbin{\tilde{\wedge}} G_2)$. Clearly G_1 and G_2 are goal formulas in the sense of Definition 2.10. Let us fix a fresh variable f^{BS}.

Following Corollary 2.14, we translate the proof Mf to the proof

$$\begin{aligned}
&\mathcal{P}' := \lambda v^{\tilde{A}}\, Mf(\lambda k_1\, \lambda k_2\, \lambda w_1^{k_1<k_2}\, \lambda w_2^{fk_1=fk_2}\, \mathcal{Q}_1 w_1(\lambda z_1^{k_1<k_2}\, \mathcal{Q}_2 w_2(\lambda z_2^{fk_1=fk_2}\, v k_1 k_2 z_1 z_2))),\\
&\mathcal{Q}_i := \lambda a_i\, \lambda b_i\, b_i a_i, \text{ for } i = 1, 2, \text{ where}\\
&a_1 : k_1 < k_2, \qquad b_1 : k_1 < k_2 \to \bot,\\
&a_2 : fk_1 = fk_2, \qquad b_2 : fk_1 = fk_2 \to \bot.
\end{aligned}$$

Finally, we need to extract from the proof of $A := \exists k_1\, \exists k_2\, (G_1 \wedge G_2)$, which is given by

$$\mathcal{P} := \lambda f\, \mathcal{P}'\, [\bot := A]\, (\lambda k_1\, \lambda k_2\, \lambda w_1^{k_1<k_2}\, \lambda w_2^{fk_1=fk_2}\, \exists^+_{k_1} k_1 (\exists^+_{k_2} k_2\, \langle w_1, w_2\rangle)).$$

Before we start the extraction, for clarity, let us denote $\mathsf{R} := \tau^\circ(A) = \mathsf{N} \times \mathsf{N}$. This type will have a special meaning, because it is the *result type*, i.e., the type of the witness, which we want to extract. Since we substitute $[\bot := A]$, this type will appear in every part of the extracted program. To simplify notation, below we will not write the substitution explicitly when we define the extracted terms below, i.e., we will write just $[\![M]\!]^\circ$ instead of $[\![M\,[\bot := A]]\!]^\circ$.

We start the extraction process bottom-up from the lemma L:

$$\begin{aligned}
&[\![M_{\mathsf{tt}}]\!]^\circ \equiv \lambda x^{\mathsf{R}}\, \lambda y^{\mathsf{R}}\, x, \qquad [\![M_{\mathsf{ff}}]\!]^\circ \equiv \lambda x^{\mathsf{R}}\, \lambda y^{\mathsf{R}}\, y,\\
&[\![C_B]\!]^\circ \equiv \lambda b\, \mathsf{Cases}^{\mathsf{R}\Rightarrow\mathsf{R}\Rightarrow\mathsf{R}}\, b\, [\![M_{\mathsf{tt}}]\!]^\circ [\![M_{\mathsf{ff}}]\!]^\circ \overset{r}{=} \mathsf{Cases}^{\mathsf{R}},\\
&[\![L]\!]^\circ \equiv \lambda f\, \lambda g\; g\, \mathsf{tt}(\lambda n_{\mathsf{tt}}^{\mathsf{N}}\, \lambda h_{\mathsf{tt}}^{\mathsf{N}\Rightarrow\mathsf{R}}\, g\, \mathsf{ff}(\lambda n_{\mathsf{ff}}^{\mathsf{N}}\, \lambda h_{\mathsf{ff}}^{\mathsf{N}\Rightarrow\mathsf{R}}\, \mathsf{Cases}(fm)(h_{\mathsf{tt}} m)(h_{\mathsf{ff}} m))),\\
&\quad m := n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}},\\
&\quad g : \mathsf{B} \Rightarrow (\mathsf{N} \Rightarrow (\mathsf{N} \Rightarrow \mathsf{R}) \Rightarrow \mathsf{R}) \Rightarrow \mathsf{R}.
\end{aligned}$$

Above we used the fact that the normal forms of $\mathsf{Cases}^{\mathsf{R}}$ and $[\![C_B]\!]^\circ$ coincide. We continue with the proofs $M, \mathcal{P}'$ and finally $\mathcal{P}$:

$$\begin{aligned}
[\![M]\!]^\circ &\equiv \lambda g^{\mathsf{N}\Rightarrow\mathsf{N}\Rightarrow\mathsf{R}}\, [\![L]\!]^\circ(\lambda b\, \lambda h^{\mathsf{N}\Rightarrow(\mathsf{N}\Rightarrow\mathsf{R})\Rightarrow\mathsf{R}}\, h0(\lambda k_1\, h(\mathsf{S}k_1)(\lambda k_2\, g k_1 k_2))),\\
[\![\mathcal{P}']\!]^\circ &\equiv \lambda g^{\mathsf{N}\Rightarrow\mathsf{N}\Rightarrow\mathsf{R}}\, [\![M]\!]^\circ\, f\, (\lambda k_1\, \lambda k_2\, g k_1 k_2)\\
&\overset{r}{=} \lambda g\, [\![M]\!]^\circ\, f\, g \overset{r}{=} [\![M]\!]^\circ\, f,\\
[\![\mathcal{P}]\!]^\circ &\equiv \lambda f\, [\![M]\!]^\circ\, f(\lambda k_1\, \lambda k_2\, (\lambda m_1\, \lambda m_2\, \langle m_1, m_2\rangle) k_1((\lambda m_2\, m_2) k_2))\\
&\overset{r}{=} \lambda f\, [\![M]\!]^\circ\, f(\lambda k_1\, \lambda k_2\, \langle k_1, k_2\rangle).
\end{aligned}$$

Having in mind Murthy's results relating A-translation and the continuation passing style, we can think of the functions of type $\tau \Rightarrow \mathsf{R}$ as continuations. This can help to explain the behaviour of the extracted program above by proceeding in order of increase of $\deg(\tau)$.

The continuations h_b in $[\![L]\!]^\circ$ can be viewed as instructions how to continue, in case we provide it with an occurrence of b after the index n_b. The body of $[\![L]\!]^\circ$ makes a case distinction on the value of fm and invokes the appropriate continuation for the index $m := n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}$. On the other hand, the continuation h in $[\![M]\!]^\circ$ expects a number n and a continuation of the type of h_b, so that h_b gives a result when invoked at any $k \geq n$ with $fk = b$. The continuation h is used in $[\![M]\!]^\circ$ two times: for 0 in order to obtain some index k_1 of b and then for $\mathsf{S}k_1$ to obtain another index $k_2 > k_1$ of b. Finally, the crucial operation of $[\![L]\!]^\circ$ is determined by the continuation parameter g. It describes how to proceed in case there is a boolean b, for which we can continue infinitely often, i.e., for any number n and a continuation of the type h_b we can produce a result. In fact, this is the same as providing g with a continuation parameter of the type of h. $[\![L]\!]^\circ$ invokes g twice for each boolean, which corresponds to the classical case distinction "some boolean must appear infinitely often". Intuitively, g collects requests for finding occurrences of the respective boolean. Since the sequence consists only of booleans, one of those requests can always be answered, depending on the sequence f.

The program $[\![M]\!]^\circ$ makes two sequential requests by invoking the continuation h twice. Hence, if the first two values in the sequence are equal, then the program will output the indices $\langle 0, 1\rangle$. However, when the first two values are different, then backtracking will occur. Moreover, the behaviour of the program will be asymmetric, since the invocations of g are not parallel, but nested within each other. Thus if the sequence starts with $\mathsf{tt}, \mathsf{ff}, b$, then the program will return the indices $\langle 0, 2\rangle$ or $\langle 1, 2\rangle$, depending on the value of b. Similarly, if the sequence starts with $\mathsf{ff}, \mathsf{tt}, \mathsf{tt}$, the indices returned will be $\langle 1, 2\rangle$. However, if the sequence starts with $\mathsf{ff}, \mathsf{tt}, \mathsf{ff}, b$. Then, the returned indices will not be $\langle 0, 2\rangle$, but $\langle 1, 3\rangle$ or $\langle 2, 3\rangle$, depending on the value of b. The reason is that the first occurrence of ff is "forgotten", since it is stored inside the continuation for the case of tt. Therefore, the program will never return indices of ff, which differ by more than 1, since an occurrence of tt will erase all previously stored occurrences of ff. The particular behaviour of the program can be demonstrated best by the normal form of $[\![\mathcal{P}]\!]^\circ$:

$$\begin{aligned}[\![\mathcal{P}]\!]^\circ \overset{r}{=} \lambda f\, \mathsf{Cases}\,(f0)\,&(\mathsf{Cases}\,(f1)\ \langle 0, 1\rangle\ \langle(\mathsf{Cases}\,(f2)\,0\,1), 2\rangle)\\ &(\mathsf{Cases}\,(f1)\,(\mathsf{Cases}\,(f2)\ \langle 1, 2\rangle\ \langle(\mathsf{Cases}\,(f3)\,1\,2), 3\rangle)).\end{aligned}$$

The same behaviour is noticed in [Urb00, Mak06, Rat10]. In all these cases it is exactly the sequentiality in which the cases of tt and ff are treated which is responsible for the asymmetry.

3.1.3 Extraction via the Dialectica interpretation

We will extract from the proof $M\,[\bot := \mathrm{F}]$. To simplify notation, we will not explicitly mention the substitution. Also, for clarity we will index the negative computational content with assumption variables instead of numbers.

We start extraction from the lemma L:

$\mathcal{P}$	$[\![\mathcal{P}]\!]^+$	$\bullet$	$[\![\mathcal{P}]\!]^-_\bullet$
L_b	ε	v_b	$n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}$
$L_1 := C_B(f(n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}))L_{\mathsf{tt}}L_{\mathsf{ff}}$	ε	v_b	$n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}$
$L_2 := \lambda n_{\mathsf{ff}}\, \lambda v_{\mathsf{ff}}\, L_1$	$\lambda n_{\mathsf{ff}}\, n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}$	v_{tt}	$n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}$
$L_3 := \lambda n_{\mathsf{tt}}\, \lambda v_{\mathsf{tt}}\, u\, \mathsf{ff}\, L_2$	$\lambda n_{\mathsf{tt}}\, x_u\, \mathsf{ff}\, [\![L_2]\!]^+$	u	$\left\langle \mathsf{ff}, [\![L_2]\!]^+ \right\rangle$
$L_4 := u\, \mathsf{tt}\, L_3$	ε	u	$\left\langle \mathsf{tt}, [\![L_3]\!]^+ \right\rangle \overset{u}{\bowtie} \left\langle \mathsf{ff}, \lambda n_{\mathsf{ff}}\, (x_u\, \mathsf{tt}\, [\![L_3]\!]^+) \sqcup n_{\mathsf{ff}} \right\rangle$
$L\ := \lambda f\, \lambda u\, L_4$	$\lambda f\, \lambda x_u\, [\![L_4]\!]^-_u$		

We only need to calculate the case distinction $\overset{u}{\bowtie}$:

$$
\begin{aligned}
&\left|\forall b\, \tilde{\exists} n\, \forall k\, (n \le k \to fk = b \to \mathrm{F})\right|^{x_u}_{\left\langle \mathsf{tt}, [\![L_3]\!]^+ \right\rangle} = \\
&= \left|\tilde{\exists} n\, \forall k\, (n \le k \to fk = \mathsf{tt} \to \mathrm{F})\right|^{x_u \mathsf{tt}}_{[\![L_3]\!]^+} \\
&= \left|\forall k\, (([\![L_3]\!]^+ m) \le k \to fk = \mathsf{tt} \to \mathrm{F})\right|^{\varepsilon}_{m}, \text{ where } m := x_u \mathsf{tt} [\![L_3]\!]^+ \\
&= ([\![L_3]\!]^+ m) \le m \to fm = \mathsf{tt} \to \mathrm{F}, \\
[\![L_4]\!]^-_u &\equiv \mathsf{Cases}\, (([\![L_3]\!]^+ m) \le m \to fm \ne \mathsf{tt})^{\mathrm{at}}\, \langle \mathsf{ff}, \lambda n_{\mathsf{ff}}\, m \sqcup n_{\mathsf{ff}} \rangle\, \left\langle \mathsf{tt}, [\![L_3]\!]^+ \right\rangle.
\end{aligned}
$$

We are ready to extract from the proof M:

$\mathcal{P}$	$[\![\mathcal{P}]\!]^+$	$\bullet$	$[\![\mathcal{P}]\!]^-_\bullet$
$M_1 := v k_1 k_2 (M_< u_2)(M_= z_1 z_2)$	ε	v	$\langle k_1, k_2 \rangle$
$M_2 := \lambda k_2\, \lambda u_2\, \lambda z_2\, M_1$	ε	v	$\langle k_1, k_2 \rangle$
$M_3 := \lambda k_1\, \lambda u_1\, \lambda z_1\, w(\mathsf{S}k_1) M_2$	ε	v	$\langle k_1, x_w(\mathsf{S}k_1) \rangle$
		w	$\mathsf{S}k_1$
$M_4 := w\, 0\, M_3$	ε	v	$\langle x_w 0, x_w(\mathsf{S}(x_w 0)) \rangle$
		w	$0 \overset{w}{\bowtie} \mathsf{S}(x_w 0)$
$M_5 := \lambda b\, \lambda w\, M_4$	$\lambda b\, \lambda x_w\, [\![M_4]\!]^-_w$	v	$[\![M_4]\!]^-_v$
$M\ := \lambda f\, \lambda v\, L f M_5$	$\lambda f\, [\![M_4]\!]^-_v \left[x_w := [\![L]\!]^+ f [\![M_5]\!]^+ \lrcorner\right]$		

The case distinction $\overset{w}{\bowtie}$ is unfolded as follows:

$$\begin{aligned}\left|\forall n\,\tilde{\exists} k\,(n \le k \mathbin{\tilde{\wedge}} fk = b)\right|_{0}^{x_w} &= \left|\tilde{\exists} k\,(0 \le k \mathbin{\tilde{\wedge}} fk = b)\right|_{\varepsilon}^{x_w 0}\\ &= 0 \le (x_w 0) \to f(x_w 0) = b \to \mathrm{F},\\ [\![M_4]\!]_w^- &\equiv \mathsf{Cases}(f(x_w 0) \ne b)^{\mathrm{at}}\,\big(x_w(\mathsf{S}(x_w 0))\big)\,(x_w 0).\end{aligned}$$

In order to understand the behaviour of the extracted program, we need to remember that positive computational content provides witnesses for the respective formula and negative computational content computes challenges for these witnesses. Thus the functions $[\![L_2]\!]^+$ and $[\![L_3]\!]^+$ are given a number n and compute an index $\ge n$ at which respectively ff and tt appear. Both functions operate under the assumption w that there is a maximal index for every boolean. Thus $[\![L_2]\!]^+$ uses the index n_{tt} after which tt is not supposed to appear, according to the assumption u_{tt}. On the other hand $[\![L_3]\!]^+$ queries to the witness x_w of w at ff to find an index n after which ff is not supposed to appear. However, x_w expects an additional parameter: a function which when given n provides a challenge for an index larger than n where ff does not appear, i.e., a function given an index larger than n where ff *does* appear. Clearly, $[\![L_2]\!]^+$ is exactly such a function. Note that $[\![L_3]\!]^+$ plays the symmetric role when w is queried at tt. On the other hand, the negative contents of both proofs L_2 and L_3 constitute exactly of the arguments used to compute the positive content. The reason is that the correctness of the positive computational content depends on the validity of the assumptions and hence the arguments to the witnesses of the assumptions are also the possible counterexamples for them. Finally, the lemma L proves arrives at a contradiction with the assumption u by using it twice. Thus it produces a two possible challenges for it and by direct case distinction on the translation of the formula of u determines which one is correct. The correct challenge for u is the correct witness for the whole lemma.

We now turn our attention to the main program $[\![M]\!]^+$. The pair of indices, which is the final witness, is also a challenge for the false assumption v that there are no two equal values in the sequence f. The terms $[\![M_3]\!]_v^-$ and $[\![M_4]\!]_v^-$ construct the witnesses for k_2 and k_1 respectively, by utilizing the witness x_w for the assumption w coming from the lemma L: there is an occurrence of a boolean b after any index n. However, since w is invoked twice (once for each witness from the pair), the correct challenge for it needs to be determined by a case distinction on the translation of the formula of w. Let us recall that $[\![L]\!]^+$ expected as a parameter a function that provides an index after which b should not appear. This is precisely the challenge for w, so $[\![M_5]\!]^+$ is a natural choice for the parameter of $[\![L]\!]^+$. Finally, $[\![M]\!]^+$ invokes $[\![L]\!]^+$ to construct the witness for w, which is then used to compute the final witness. In short, the interaction between the programs $[\![M]\!]^+$ and $[\![L]\!]^+$ consists of mutual feedback, since the challenges in M are witnesses for L and vice versa.

The normal form of the program $[\![M]\!]^+$ is quite long, so we rather display a shorter term, which can be proved to be extensionally equal to $[\![M]\!]^+$:

$$\begin{aligned}
[\![M]\!]^+ \overset{\mathsf{BS}\Rightarrow\mathsf{N}\times\mathsf{N}}{=} \lambda f\, &\textbf{let } h_1 := \lambda n\, \mathsf{Cases}\,(fn)\,0\,(\mathsf{S}n) \textbf{ in} \\
&\textbf{let } h_2 := \lambda n\, \mathsf{Cases}\,(fn)\,(\mathsf{S}n)\,0 \textbf{ in} \\
&\textbf{let } n_1 := h_2(h_1 0) \textbf{ in} \\
&\textbf{let } n_2 := h_1 n_1 \textbf{ in} \\
&\textbf{let } h := \lambda n\, \mathsf{Cases}\,(f(n_1 \sqcup n_2))\,(n \sqcup (h_1 n))\,(n_1 \sqcup n) \textbf{ in} \\
&\langle h0, h(\mathsf{S}(h0))\rangle\,.
\end{aligned}$$

The term above has been obtained by manual simplifications of the normal form of $[\![M]\!]^+$ by introducing appropriate let definitions and applying the following reductions:

$$\begin{aligned}
\max 0\, n &\mapsto n, & n_1 \le \max n_1 n_2 &\mapsto \mathsf{tt}, \\
\max n\, 0 &\mapsto n, & n_2 \le \max n_1 n_2 &\mapsto \mathsf{tt}, \\
\max n\, n &\mapsto n, & n \le n &\mapsto \mathsf{tt}, \\
b = \mathsf{tt} &\mapsto b, & (\neg\neg \mathrm{at}(b))^{\mathrm{at}} &\mapsto b,
\end{aligned}$$

3.1.4 Comparison

The programs $[\![\mathcal{P}]\!]^\circ$ and $[\![M]\!]^+$, extracted respectively with refined A-translation and the Dialectica interpretation, are quite different, even though obtained from the same proof M. What is surprising is that they happen to be extensionally equal, i.e., produce equal witnesses on equal sequences. In fact, the returned pair of indices depends only on the first four elements in the sequence. The behaviour of the programs is summarised in Table 3.1.

f	$[\![\mathcal{P}]\!]^\circ f$
ff, ff, ...	$\langle 0, 1\rangle$
tt, tt, ...	$\langle 0, 1\rangle$
tt, ff, ff, ...	$\langle 1, 2\rangle$
ff, tt, tt, ...	$\langle 1, 2\rangle$
tt, ff, tt, ...	$\langle 0, 2\rangle$
ff, tt, ff, tt ...	$\langle 1, 3\rangle$
ff, tt, ff, ff ...	$\langle 2, 3\rangle$

Table 3.1: Witnesses produced by programs extracted from Stolzenberg's example

The asymmetry of the results can be traced back to the proof of L by using the assumption u first for $\mathtt{tt}$ and then for $\mathtt{ff}$. Effectively, the value $\mathtt{tt}$ obtains a "higher priority": in both programs the functions computing indices of $\mathtt{ff}$ are always invoked from functions computing indices of $\mathtt{tt}$.

Both programs reflect the use of classical logic by some form of backtracking. Generally, the backtracking is triggered by invoking a functional parameter with two different values: $\mathtt{tt}$ and $\mathtt{ff}$. However, in $[\![\mathcal{P}]\!]^{\circ}$ the functional parameters are continuations, in the sense that the invocations are always tail-recursive (cf. [Rat10]). This is not the case in $[\![M]\!]^{+}$, where functional parameters compute challenge candidates whose validity controls the backtracking process. Both programs use similar case distinctions with quite different origins: while in $[\![L]\!]^{\circ}$ the only case distinction comes from the *proof* of C_B, in both $[\![L]\!]^{+}$ and $[\![M]\!]^{+}$, the case distinction is the quantifier-free translation of the *formula* proved by L.

Stolzenberg's example is not parametrised by a number, hence theoretically both programs have constant time complexity. However, it is easy to see that $[\![M]\!]^{+}$ is not only larger, but also less efficient. The reason is that the subterms of ground type m and x_w0 are repeated several times throughout the programs $[\![L]\!]^{+}$ and $[\![M]\!]^{+}$ and will be hence redundantly evaluated to the same number more than once under any reduction strategy. The problem can be partially remedied by using a let construct in the case distinction operator $\overset{u}{\bowtie}$, but will not be solved completely; x_w0 will still be repeated in the term $[\![M_4]\!]^{-}_{w}$. The next case studies will help to outline the nature of this inefficiency and how it can be solved.

3.2 Integer root

One possible application of methods for extraction from classical proofs is to obtain a valid program from a non-constructive proof of existence, which is presumably easier to provide than an explicit (constructive) proof. Stolzenberg's example demonstrated how programs obtained from a proof that makes non-trivial use of classical logic, are not necessarily optimal. The present example will investigate the behaviour of programs extracted from classical proofs, which are essentially constructive, i.e., prove contradiction from a false assumption without using it more than once.

We will use the "integer root" example, presented in [BS95, Ber95] and later treated also in [Mak06]. The example can be stated as follows.

Proposition 3.4. *Let $f : \mathsf{N} \Rightarrow \mathsf{N}$ be an unbounded function, i.e., there exists a function $g : \mathsf{N} \Rightarrow \mathsf{N}$, such that $\forall n\,(f(gn) > n)$. Then for every $m \geq f0$ there is an n, such that $fn \leq m < f(n+1)$.*

Proof. We assume that there is no such n, i.e., $\forall n\,(fn \le m \to f(n+1) \le m)$. By induction we can prove that $\forall n\,(fn \le m)$, and by setting $n := gm$ we arrive at a contradiction. $\square$

3.2.1 Proof formalisation

Let us denote the type of sequences of natural numbers as $\mathsf{NS} := \mathsf{N} \Rightarrow \mathsf{N}$. The existence of an integer root can be formalised as follows:

$$\forall f^{\mathsf{NS}}\, \forall g^{\mathsf{NS}}\, \forall m^{\mathsf{N}}\, (\forall n^{\mathsf{N}}\,(f(gn) > n) \to \neg(f0 > m) \to \tilde{\exists} n^{\mathsf{N}}\,(\neg(fn > m) \mathbin{\tilde{\wedge}} f(\mathsf{S}n) > m)).$$

The proof of Proposition 3.4 can be expressed by the following proof term:

$$\begin{aligned} M := \lambda f\, \lambda g\, \lambda m\, \lambda u^{\forall n\,(f(gn)>n)} \lambda v^{\neg(f0<m)} \lambda w^{\forall n\,(\neg(fn>m)\to\neg(f(\mathsf{S}n)>m))} \\ \mathsf{Ind}_{n,\neg(fn>m)}\,(gm)\,v\,w\,(um). \end{aligned}$$

The formalisation is particularly simple, because we have chosen to use only the relation $>$ and defined $\le$ as its negation. Moreover, a constructive proof of this statement would be much more involved, as it would require an additional lemma [BS95].

3.2.2 Extraction via refined A-translation

This example has already been treated in [BS95] with refined A-translation. M proves a formula of the form $D_1 \to D_2 \to \tilde{\exists} n\,(G_1 \mathbin{\tilde{\wedge}} G_2)$, where $D_1 := \forall n\,(f(gn) > n)$ and $D_2 := \neg(f0 > m)$ are definite formulas, while $G_1 := \neg(fn > m)$ and $G_2 := f(\mathsf{S}n) > m$ are goal formulas.

By Corollary 2.14, we need to extract from the translated proof

$$\begin{aligned} \mathcal{P} &:= \lambda u_1^{D_1^F}\, \lambda u_2^{D_2^F}\, \mathcal{P}'\,[\bot := \exists n\,(G_1 \wedge G_2)]\, u_1 u_2 (\lambda n\, \lambda v_1^{G_1^F}\, \lambda v_2^{G_2^F}\, \exists^+ n\,\langle v_1, v_2\rangle), \text{ where} \\ \mathcal{P}' &:= \lambda u_1^{D_1^F}\, \lambda u_2^{D_2^F}\, \lambda v^{\forall n\,(\vec{G}^{\mathrm{F}} \to \bot)}\, M f g m (\mathcal{Q}_1' u_1)(\mathcal{Q}_2' u_2) \\ &\qquad\qquad (\lambda n\, \lambda w_1^{G_1}\, \lambda w_2^{G_2}\, \mathcal{Q}_1'' w_1 (\lambda z^{G_1^F}\, \mathcal{Q}_2'' w_2 (vnz)), \\ \mathcal{Q}_1' &:= \lambda u_1^{D_1}\, u_1, \\ \mathcal{Q}_2' &:= \lambda u_2^{D_2^F}\, \lambda v_2^{f0>m}\, \bot^+(u_2 v_2), \\ \mathcal{Q}_1'' &:= \mathsf{CD}_{fn>m,\bot}, \\ \mathcal{Q}_2'' &:= \lambda w_2^{G_2}\, \lambda v_2^{G_2\to\bot}\, v_2 w_2. \end{aligned}$$

To simplify notation, in the following we will not explicitly denote the substitution $[\bot := \exists n\,(G_1 \wedge G_2)]$ in the proofs below.

$$\begin{aligned}
[\![\mathcal{Q}_1']\!]^\circ &\equiv \varepsilon, \qquad [\![\mathcal{Q}_2']\!]^\circ \equiv \square^{\mathsf{N}} \equiv 0, \qquad [\![\mathcal{Q}_2'']\!]^\circ \equiv \lambda k\, k,\\
[\![\mathcal{Q}_1'']\!]^\circ &\equiv \mathsf{Cases}^{\mathsf{N}\Rightarrow\mathsf{N}\Rightarrow\mathsf{N}}(fn > m)(\lambda x\, \lambda y\, x)(\lambda x\, \lambda y\, y) \overset{r}{=} \mathsf{Cases}^{\mathsf{N}}(fn > m)\\
[\![Mfgm]\!]^\circ &\overset{r}{=} \mathcal{R}_{\mathsf{N}}^{\mathsf{N}}(gm),\\
[\![\mathcal{P}']\!]^\circ &\equiv \lambda h^{\mathsf{N}\Rightarrow\mathsf{N}}\, \mathcal{R}_{\mathsf{N}}(gm)0(\lambda n\, \lambda p\, \mathsf{Cases}(fn > m)p(hn)),\\
[\![\mathcal{P}]\!]^\circ &\equiv [\![\mathcal{P}']\!]^\circ(\lambda n\, n) \overset{r}{=} \mathcal{R}_{\mathsf{N}}(gm)0(\lambda n\, \lambda p\, \mathsf{Cases}(fn > m)pn),
\end{aligned}$$

The program $\mathcal{P}$ is already in normal form. It performs a linear search for a number n with $fn \leq m$ starting from gm down to 0 and returns the first one found.

3.2.3 Extraction via the Dialectica interpretation

Let $M_1 := \mathsf{Ind}_{n,\neg(fn>m)}\,(gm)\,v\,w(um)$. As above, we will not explicitly denote the substitution $[\bot := \mathrm{F}]$. Then

$$\begin{aligned}
[\![M]\!]^+ &\equiv \lambda f\, \lambda g\, \lambda m\, \left\langle [\![M_1]\!]_u^-, [\![M_1]\!]_w^- \right\rangle\\
&\equiv \lambda f\, \lambda g\, \lambda m\, \left\langle m, \mathcal{R}_{\mathsf{N}}(gm)\,\square^{\mathsf{N}}\,(\lambda n\, \lambda p\, (p \overset{w}{\bowtie} n))\right\rangle, \text{ where}\\
p \overset{w}{\bowtie} n &\equiv \mathsf{Cases}\big(T_\to(fp \leq m)(m \geq f(\mathsf{S}p))\big)np.
\end{aligned}$$

The program $[\![M]\!]^+$ is very similar to $[\![\mathcal{P}]\!]^\circ$, however, there are two prominent differences. First of all, the Dialectica interpretation extracts more information from the proof. Apart from the program $[\![M_1]\!]_w^-$, which computes the counterexample for w, and hence the witness for $\tilde{\exists}n$, we also extract the term $[\![M_1]\!]_u^-$, which is a counterexample for the assumption u, stating that g bounds f at every n. The underlying reason is that classically we can read Proposition 3.4 as "For a function f if some number $m \geq f0$ has no integer root, then f is bounded by some number M". Then from the Dialectica interpretation of the proof we can see that m is a witness for M, which, in fact, does not depend on g. We could obtain the same witness by refined A-translation, but we would need to rearrange the formalisation of M and then the translated proof $\mathcal{P}$ would be completely different. In contrast, via the Dialectica interpretation we obtain both witnesses, even if we are only interested in one of them.

The second difference between the two programs $[\![M]\!]^+$ and $[\![\mathcal{P}]\!]^\circ$ lies in the recursive processes generated by them. While in $[\![\mathcal{P}]\!]^\circ$ the recursion starts from gm and stops unfolding as soon as an integer root is found, in $[\![M]\!]^+$ the case distinction involves the variable p, which corresponds to a recursive call. Hence, the recursion in $[\![M]\!]^+$ always unfolds to 0 and then the search starts from 0 up towards gm. Once an integer root k is found, then every subsequent case distinction for $n > k$ will always evaluate to ff and k will be the finally computed witness. However, all these case distinctions for $n \in (k; gm]$ will be redundant, because they will reconfirm, what is already known: that k is the witness. Thus $[\![M]\!]^+$ and $[\![\mathcal{P}]\!]^\circ$ return the *smallest* and the *largest* integer root of f in the interval $[0; gm]$, respectively. Moreover, the two programs have the same worst time complexity $O(gm)$. However, $[\![M]\!]^+$ can be noticeably slower on average, since it will perform always exactly gm steps, while $[\![\mathcal{P}]\!]^\circ$ will only perform $gm - k$ number of steps, where k is the largest integer root of f, which is smaller than gm. Hence, if g produces a good approximation of an integer root on average, the program $[\![\mathcal{P}]\!]^\circ$ will have better average time complexity.

There are two easy solutions, which can improve the program $[\![M]\!]^+$. The first one is to use the case distinction operator with reversed arguments:

$$n \overset{w}{\bowtie} p \equiv \mathsf{Cases}\big(T_\to(fn \le m)(m \ge f(\mathsf{S}n))\big)pn.$$

This change is sound, because in Theorem 2.21 the order of arguments is not important. It is easy to see that in this way the program becomes almost the same as $[\![\mathcal{P}]\!]^\circ$, the only difference being that we still perform two comparisons instead of one. The modified program will now return the largest integer root.

The other solution is to introduce a boolean flag, which “remembers” that a counterexample is found and avoids further case distinctions. The altered program will look as follows:

$$\begin{aligned}[\![M_1]\!]^-_w \equiv \mathcal{R}^{\mathsf{N}\times\mathsf{B}}_{\mathsf{N}}(gm)\,\langle 0,\mathsf{ff}\rangle(\lambda n\,\lambda p\,(\mathsf{Cases}(p_\lrcorner)p\\ (\mathsf{Cases}\big(T_\to(f(p_\llcorner) \le m)(m \ge f(\mathsf{S}(p_\llcorner)))\big)\,\langle n,\mathsf{ff}\rangle\,\langle p_\llcorner,\mathsf{tt}\rangle)))\end{aligned}$$

Note that p is now a pair $\big\langle n^{\mathsf{N}}, b^{\mathsf{B}}\big\rangle$, where if b is tt, then n has been already verified to be a witness. The program will retain its original behaviour computing the smallest integer root, but it will skip the unnecessary case distinctions involving n, by reducing them to verifying the value of the boolean flag b. This change can also be proved to be sound, but the proof is more involved, as it requires an additional statement about the soundness of the flag b.

The first solution seems easier and more natural, but we will later demonstrate that it does not improve the efficiency in more general cases, in particular, when the induction formula is not quantifier-free. However, we will see that in cases where the induction formula requires witnesses but does not require challenges, the second solution still applies.

3.3 Infinite Pigeonhole Principle

The third example which we will consider is an extension of the Pigeonhole Principle, known as the Infinite Pigeonhole Principle:

Theorem 3.5 (Infinite Pigeonhole Principle)**.** *In every infinite sequence of finitely many colours, there is a colour which occurs infinitely often.*

Proof. Induction on the number of colours. For 0 and 1 colours we have nothing to prove. Assume that we have an infinite sequence of $n + 1$ colours and consider cases on the statement "The colour n appears infinitely often." In case of a positive answer the claim is proved. Assume that the colour n occurs only a finite number of times, then there is an index k after which the sequence does not contain the colour n. Hence the subsequence starting at k contains n colours only and by induction hypothesis there must be a colour in it, which occurs infinitely often. $\square$

The proof presented above is clearly non-constructive as it contains an undecidable case distinction on whether a certain colour occurs infinitely often. The Infinite Pigeonhole Principle can be viewed as a special case of the Infinite Ramsey Theorem. The constructive meaning of both principles has been investigated by several authors, including Veldman and Bezem [VB93], Coquand [Coq94], Tao [Tao07], Gaspar and Kohlenbach [GK10]. There is an important difference between previous work and the present analysis of the principle. In order to obtain the representation of the whole infinite subsequence of the same colour, a limited form of the classical Axiom of Choice is required, which is known as the Axiom of Dependent Choice. The computational meaning of this axiom is Spector's bar recursion [Spe62] and constructivisations of the Infinite Pigeonhole Principle contain some similar form of recursion. Here we analyse a simpler case, in which the infinitely many occurrences of some colour c are rephrased as *arbitrarily high occurrences*, i.e., for every number n there is an index $m \geq n$ of the colour c. This formulation avoids the use of choice principles and allows us to concentrate on the computational contribution of pure classical logic. The results in this section are based on joint work with Diana Ratiu and are published as [RT09].

3.3.1 Proof formalisation

In the following we will denote the type of sequences of natural numbers as NS. We will denote the maximum of two numbers a and b as $a \sqcup b$. The statement of the Infinite Pigeonhole Principle can be formalised as:

$$\forall r^{\mathsf{N}} \forall f^{\mathsf{NS}} \left(\forall n^{\mathsf{N}} (fn < r) \to \tilde{\exists} q^{\mathsf{N}} \forall n^{\mathsf{N}} \tilde{\exists} m^{\mathsf{N}} (n \leq m \mathbin{\tilde{\wedge}} fm = q)\right).$$

Unfortunately, the formula above cannot be proved by induction on r in MA^ω, which we need in order to apply refined A-translation. The reason is that we need to derive $\bot$ using the premise $\forall n\,(fn < r)$, which does not involve $\bot$. Thus we consider a slightly stronger formulation, in which the atom in the premise is double negated:

$$\forall r^{\mathsf{N}}\,\forall f^{\mathsf{NS}}\,(\forall n^{\mathsf{N}}\,(\neg\neg fn < r) \to \tilde{\exists} q^{\mathsf{N}}\,\forall n^{\mathsf{N}}\,\tilde{\exists} m^{\mathsf{N}}\,(n \leq m \,\tilde{\wedge}\, fm = q)).$$

Note that after substituting $[\bot := \mathrm{F}]$, the two formulas become equivalent, and thus their witnesses coincide.

Let us elaborate on the proof of the Infinite Pigeonhole Principle formulated as above. We proceed by induction on r. For the base case we prove $\bot$ by using the premise with $\bot^+$, as $fn < 0$ is in fact F. For the step case we assume the induction hypothesis and then assume that f is coloured with $r+1$ colours and also that there is no colour which appears infinitely often. We apply the latter assumption to the last colour r to obtain an index n after which it does not appear. Then we use the induction hypothesis on the sequence $f' = \lambda n'\, f(n \sqcup n')$, which is equal to the sequence f with the first n elements overwritten by the colour fn. We can prove that f' is a sequence of r colours and thus it has a colour q appearing infinitely often. But then the colour q should also appear infinitely often in the original sequence f, which leads to a contradiction.

The formal proof is given below:

$$\begin{aligned}
L &:= \lambda r\,\mathsf{Ind}\,r\,L_0\,(\lambda r\,\lambda p\,L_{\mathsf{S}}),\ \text{ where}\\
L_0 &:= \lambda f\,\lambda u^{\forall n\,(\neg\neg fn<0)}\,\lambda v\,u0\bot^+,\\
L_{\mathsf{S}} &:= \lambda f\,\lambda u_1\,\lambda u_2\,u_2 r(\lambda n_1\,\lambda v_1\,p(\lambda n_2\,f(n_1 \sqcup n_2))K_1K_2),\\
K_1 &:= \lambda n_2\,\lambda z^{\neg f(n_1\sqcup n_2)<r}\,u_1(n_1 \sqcup n_2)(\lambda a^{f(n_1\sqcup n_2)<\mathsf{S}r}\,L_{<}az(v_1(n_1 \sqcup n_2)L_{\sqcup})),\\
K_2 &:= \lambda q\,\lambda w\,u_2 q(\lambda n_2\,\lambda v_2\,wn_2(\lambda m\,\lambda a^{n_2\leq m}\,v_2(n_1 \sqcup m)(L_{\leq}a))),\\
u_1 &: \forall n\,(\neg\neg fn < \mathsf{S}r),\\
u_2 &: \forall q\,\tilde{\exists} n\,\forall m\,(n \leq m \to fm = q \to \bot),\\
v_i &: \forall m\,(n_i \leq m \to fm = q \to \bot)\ \text{for i} = 1,2,\\
w &: \forall n\,\tilde{\exists} m\,(n \leq m \,\tilde{\wedge}\, f(n_1 \sqcup m) = q),\\
L_{\sqcup} &: n_1 \leq n_1 \sqcup n_2,\\
L_{<} &: f(n_1 \sqcup n_2) < \mathsf{S}r \to \neg(f(n_1 \sqcup n_2) < r) \to \neg\neg(f(n_1 \sqcup n_2) = r),\\
L_{\leq} &: n_2 \leq m \to n_2 \leq n_1 \sqcup m.
\end{aligned}$$

In order to analyse the computational meaning of the Infinite Pigeonhole Principle, we will consider a Π^0_2 corollary, which we refer to as the Unbounded Pigeonhole Principle.

Corollary 3.6 (Unbounded Pigeonhole Principle)**.** *In every infinite sequence of finitely many colours, there are at least $n+1$ occurrences of the same color for any given n.*

Formally, we will show that the following statement holds:

$$\forall r^{\mathsf{N}}\,\forall f^{\mathsf{NS}}\,\Big(\forall n^{\mathsf{N}}\,(\neg\neg fn<r)\to\forall n^{\mathsf{N}}\,\tilde{\exists} l^{\mathsf{L(N)}}\,\big(\,|l|=\mathsf{S}n\,\tilde{\wedge}\,\mathrm{Decr}(l,n)\,\tilde{\wedge}\,\mathrm{Same}(l,n)\big)\Big),\text{where}$$

$$\begin{aligned}\mathrm{Decr}(l,n)\ &:=\forall k\,(k<n\to l_{\mathsf{S}k}<l_k),\\ \mathrm{Same}(l,n)\ &:=\forall k\,(k<n\to fl_k=fl_{\mathsf{S}k}).\end{aligned}$$

$\mathrm{Decr}(l,n)$ states that a list l of length $n+1$ is strictly decreasing, which is a technically convenient way to state that it consists of different numbers. $\mathrm{Same}(l,n)$ states that l contains indices of the same colour in f.

Although the claim can be proved by an explicit construction, we can easily derive its classical version by applying the Infinite Pigeonhole Principle: since there is a colour, which occurs infinitely often, we can just take its first n occurrences.

We will prove the Unbounded Pigeonhole Principle in two steps. First, we use induction on n and the Infinite Pigeonhole Principle to show that

$$\forall r^{\mathsf{N}}\,\forall f^{\mathsf{NS}}\,\Big(\forall n^{\mathsf{N}}\,(\neg\neg fn<r)\to\forall n^{\mathsf{N}}\,\tilde{\exists} q^{\mathsf{N}},l^{\mathsf{L(N)}}\,\big(\,|l|=\mathsf{S}n\,\tilde{\wedge}\,\mathrm{Decr}(l,n)\,\tilde{\wedge}\,\mathrm{Col}(q,l,n)\big)\Big),$$
$$\text{where }\mathrm{Col}(q,l,n):=\forall k\,(k<\mathsf{S}n\to fl_k=q).$$

Then, since we are not interested in the colour q, but only in the list l, we will show that

$$\forall q\,\big(\mathrm{Col}(q,l,n)\to\mathrm{Same}(l,n)\big).$$

Combining these two steps will prove the claim. The formal proof is presented below:

$$\begin{aligned}
M &:= \lambda r\,\lambda f\,\lambda z^{\forall n\,(\neg\neg fn<r)}\,\lambda n\,\lambda v\,Lrfz\\
&\qquad\qquad (\lambda q\,\lambda w\,M_1(\lambda l\,\lambda v_0^{|l|=\mathsf{S}n}\,\lambda v_1^{\mathrm{Decr}(l,n)}\,\lambda v_2^{\mathrm{Col}(q,l,n)}\,vlv_0v_1(M_2qv_2))),\ \text{where}\\
v &: \forall l\,(|l|=\mathsf{S}n\to\mathrm{Decr}(l,n)\to\mathrm{Same}(l,n)\to\bot),\\
w &: \forall n\,\tilde{\exists} m\,(n\le m\to fm=q),\\
M_1 &:= \mathsf{Ind}\,n\,M_0\,(\lambda n\,\lambda p\,M_{\mathsf{S}}),\\
p &: \tilde{\exists} l\,\big(\,|l|=\mathsf{S}n\,\tilde{\wedge}\,\mathrm{Decr}(l,n)\,\tilde{\wedge}\,\mathrm{Col}(q,l,n)\big),\\
M_0 &:= \lambda u_0\,w0\big(\lambda m\,\lambda w_1^{0\le m}\,\lambda w_2^{fm=q}\,u_0(m{:})\mathsf{AxT}(\lambda k\,\mathsf{efq})(\lambda k\,w_2)\big),\\
M_{\mathsf{S}} &:= \lambda u_{\mathsf{S}n}\,p\Big(\lambda l\,\mathsf{Ind}\,l\,\mathsf{efq}\,\big(\lambda x\,\lambda l\,\lambda p'\,\lambda v_0^{|l|=\mathsf{S}n}\,\lambda v_1^{\mathrm{Decr}(x\,::\,l,n)}\,\lambda v_2^{\mathrm{Col}(q,x\,::\,l,n)}\\
&\qquad\qquad w(\mathsf{S}x)(\lambda m\,\lambda w_1^{\mathsf{S}x\le m}\,\lambda w_2^{fm=q}\,u_{\mathsf{S}n}(m::x::l)v_0M'_{\mathsf{S}}M''_{\mathsf{S}}))\Big),
\end{aligned}$$

$$\begin{aligned}
u_i &: \forall l \, \big(|l| = \mathsf{S}i \to \mathrm{Decr}(l,i) \to \mathrm{Col}(q,l,i) \to \bot \big),\\
M'_{\mathsf{S}} &:= \lambda k \, \mathsf{Ind}\, k \, (\lambda z^{0<\mathsf{S}k} \, M_{\mathsf{S}\le} w_1) \, (\lambda k \, \lambda p_k \, v_1 k),\\
M''_{\mathsf{S}} &:= \lambda k \, \mathsf{Ind}\, k \, (\lambda z^{0<\mathsf{S}k} \, w_2) \, (\lambda k \, \lambda p_k \, v_2 k),\\
M_{\mathsf{S}\le} &: \mathsf{S}x \le m \to x < m,\\
M_2 &:= \lambda q \, \lambda u^{\mathrm{Col}(q,l,n)} \, \lambda k \, \lambda z^{k<n} \, M_{=}(uk(M_{<}z))(u(\mathsf{S}k)z),\\
M_{=} &: l_k = q \to l_{\mathsf{S}k} = q \to l_k = l_{\mathsf{S}k},\\
M_{<} &: k < n \to k < \mathsf{S}n.
\end{aligned}$$

Note that only the main Ind in M_1 is a true use of induction, the other three do not ever use the induction hypothesis and are essentially only case distinctions.

3.3.2 Extraction via refined A-translation

We will apply Corollary 2.14 with $D := \forall n \, (\neg\neg fn < r)$ and $G_1 := |l| = \mathsf{S}n$, $G_2 := \mathrm{Decr}(l,n)$, $G_3 := \mathrm{Same}(l,n)$. We will extract from the translated proof

$$\begin{aligned}
\mathcal{P} &:= \lambda r \, \lambda f \, \lambda n \, \lambda u^{D^F} \, M\big[\bot := \exists l \, \vec{G}\big] M r f (\mathcal{Q}u) n (\lambda l \, \lambda v_1^{G_1} \, \lambda v_2^{G_2} \, \lambda v_3^{G_3} \, \exists^+ l \, \langle v_1, \langle v_2, v_3 \rangle\rangle),\\
\mathcal{Q} &:= \lambda u^{D^F} \, \lambda n \, \lambda v^{fn<r\to\bot} \, \mathsf{CD}_{fn<r,\bot} v (\lambda w^{fn<r\to\mathrm{F}} \, \bot^+(unw)).
\end{aligned}$$

Note that $G_i^F = G_i$ for all $i = 1,2,3$, thus the translation only involves the formula D. As before, the substitution $\big[\bot := \exists l \, \vec{G}\big]$ will be implicit. We start the extraction with L, which is the proof of the Infinite Pigeonhole Principle. For clarity, we will denote the type of the final result (i.e., l) by R. For g^{NS} we will also use the notation $g\lceil n := \lambda k \, g(n \sqcup k)$.

$$\begin{aligned}
[\![L_<]\!]^\circ &\equiv \mathsf{Cases}(f(n_1 \sqcup n_2) < r),\\
[\![K_1]\!]^\circ &\equiv \lambda n_2 \, \lambda z^{\mathsf{R}} \, \mathsf{FC}(n_1 \sqcup n_2)\Big(\mathsf{Cases}(f(n_1 \sqcup n_2) < r) z \big(\mathsf{SE}(n_1 \sqcup n_2)\big)\Big),\\
[\![K_2]\!]^\circ &\equiv \lambda q \, \lambda \mathsf{IS} \;\; \mathsf{IMS}\, q (\lambda n_2 \, \lambda \mathsf{SE} \, \mathsf{IS} n_2 (\mathsf{SE}\lceil n_2)),\\
[\![L_0]\!]^\circ &\equiv \lambda f \, \lambda \mathsf{FC} \, \lambda \mathsf{IMS} \;\; \mathsf{FC}\, 0 \, \square^{\mathsf{R}},\\
[\![L_{\mathsf{S}}]\!]^\circ &\equiv \lambda f \, \lambda \mathsf{FC} \, \lambda \mathsf{IMS} \;\; \mathsf{IMS} r (\lambda n_1 \, \lambda \mathsf{SE} \;\; p(f\lceil n_1)[\![K_1]\!]^\circ [\![K_2]\!]^\circ),\\
[\![L]\!]^\circ &\equiv \lambda r \, \mathcal{R}_{\mathsf{N}} \, r \, [\![L_0]\!]^\circ (\lambda r \, \lambda p \, [\![L_{\mathsf{S}}]\!]^\circ),\\
[\![M_0]\!]^\circ &\equiv \lambda \mathsf{ML} \;\; \mathsf{IS}\, 0 (\lambda m \, \mathsf{ML}(m{:})),\\
[\![M_{\mathsf{S}}]\!]^\circ &\equiv \lambda \mathsf{ML} \;\; p(\lambda l \, \mathcal{R}_{\mathsf{L(N)}} \, l \;\; \square^{\mathsf{R}} \, (\lambda x \, \lambda l \, \lambda p' \, \mathsf{IS} \, (\mathsf{S}x)(\lambda m \, \mathsf{ML}(m :: x :: l)))),\\
[\![M]\!]^\circ &\equiv \lambda r \, \lambda f \, \lambda \mathsf{FC} \, \lambda n \, \lambda \mathsf{ML} \;\; [\![L]\!]^\circ r f \mathsf{FC}\big(\lambda q \, \lambda \mathsf{IS} \, \mathcal{R}_{\mathsf{N}} \, n [\![M_0]\!]^\circ (\lambda n \, \lambda p \, [\![M_{\mathsf{S}}]\!]^\circ) \mathsf{ML}\big),\\
[\![\mathcal{Q}]\!]^\circ &\equiv \lambda n \, \lambda z^{\mathsf{R}} \, \mathsf{Cases}\, (fn < r) \, z \, \square^{\mathsf{R}},\\
[\![\mathcal{P}]\!]^\circ &\equiv \lambda r \, \lambda f \, \lambda n \, [\![M]\!]^\circ r f [\![\mathcal{Q}]\!]^\circ n (\lambda l \, l).
\end{aligned}$$

where the used function abbreviations are derived from their corresponding formulas, as displayed in Table 3.2.

Formula	Specification	Input	Output
$\forall n\,(\neg\neg fn < r)$	FC Finitely Coloured	n, R	R
$\forall m\,(n \leq m \to \neg(fm = q))$	SE Sequence Extension	m	R
$\forall n\,\tilde{\exists} m\,(n \leq m \mathbin{\tilde{\wedge}} fm = q)$	IS Infinite Sequence	n, SE	R
$\forall q\,\tilde{\exists} n\,\forall m\,(n \leq m \to \neg(fm = q))$	IMS Infinite Monochromatic Sequence	q, IS	R
$\forall l\,\big(\lvert l\rvert = \mathsf{S}n \to \mathrm{Decr}(l,n) \to \neg\mathrm{Same}(l,n)\big)$	ML Monochromatic List	l	R

Table 3.2: A-Translation computational types

The program $[\![\mathcal{P}]\!]^\circ$ follows closely the modular structure of the proof. However, strictly speaking it is not modular, because the type R appearing in $[\![L]\!]^\circ$ is external to the Infinite Pigeonhole Principle and depends on Corollary 3.6 from which we extract. If we wanted to extract from a different proof using the Infinite Pigeonhole Principle, then $[\![L]\!]^\circ$ would appear in the same shape, but the type R would be different. One solution to preserve modularity involves extending the modified realisability interpretation to predicate variables in the style of Berger's negative realisability [Ber95]. Then we would be able to extract a program involving a type variable R, i.e., a *polymorphic program* which can be reused across different corollaries of the Infinite Pigeonhole Principle.

In order to understand the behaviour of the program, let us first explain the role of the function variables in Table 3.2. All of them are continuations and always return a final result of type R.

- FC is presented an index n and a candidate for the final result, which is valid if the colour of f at n is below r, i.e., it doesn't violate the assumption that f contains no more than r colours;
- SE returns a final result when presented with an index m, which is an extension of the monochromatic sequence of colour q, i.e., an index after n, at which the colour q is expected to occur;
- IS is expected to be able to extend any sequence of a colour q, i.e., when presented with an index n it should compute an index $m \geq n$ of the colour q and use its second parameter SE to obtain the final result;

- IMS is provided with a monochromatic sequence in terms of a colour q and a continuation IS and is expected to apply IS to a suitable argument of the type of SE so that it produces the final result;
- ML is given a list l of length $n+1$ of different indices of the same colour and should produce the final result. Clearly, when n equals to the input parameter to the whole program, then ML should be the identity function.

The program $[\![L]\!]^\circ$ recurses on r and is provided with three additional parameters: the sequence f and the continuations FC and IMS. The case of $r = 0$ is impossible, so we provide an arbitrary witness $\square^{\mathsf{R}}$ to the continuation FC. In the case of $r+1$ colours, the program attempts to provide IMS with an infinite sequence of colour r. Whenever there is a request to find an index after n_1 of colour r, the program assumes that the colour r does not occur after n_1 and initiates a recursive call for f, where the first n_1 colours are overwritten by fn_1. The programs $[\![K_1]\!]^\circ$ and $[\![K_2]\!]^\circ$ take the role of FC and IMS respectively. $[\![K_1]\!]^\circ$ is a modification of the continuation FC by considering the question "is the colour at index $n_1 \sqcup n_2$ smaller than r". If the answer is "yes", then the assumption that the colour r does not appear after n_1 has not failed and we return the given result candidate z. However, if the answer is "no", by the assumption that there are $r+1$ colours the colour at the index $n_1 \sqcup n_2$ must be r, hence it is provided to the continuation SE. The program $[\![K_2]\!]^\circ$ corresponds to the case when the recursive call obtains a sequence of colour q provided by the continuation IS. The sequence is fed back to the parameter IMS with the modification that any obtained SE is modified to take only indices larger than n_2.

The program $[\![M]\!]^\circ$ expects continuations FC and ML; the former is directly passed to $[\![L]\!]^\circ$, while the latter is used to accumulate the final result. The continuation parameter IMS of $[\![L]\!]^\circ$ is defined by recursion on n. In the case of $n = 0$ the continuation IS for 0 is used to find the first index m of the colour q, and the singleton list m: is provided as a witness to the continuation ML. In case we need a list of length $n+2$, we first recursively construct a list of length $n+1$. This list has to be of the form $x :: l$; the nil case is discarded by producing an arbitrary result $\square^{\mathsf{R}}$. The continuation IS is used for $x+1$ to produce an index m of colour q that is strictly larger than x; it is then put in front of the already obtained list and plugged back into the continuation ML. Finally, the program $[\![\mathcal{P}]\!]^\circ$ combines everything together by instantiating FC with the correctness guard $[\![\mathcal{Q}]\!]^\circ$ and ML with the identity continuation.

In order to understand the operational semantics of the obtained program, we should note that both recursions on r and n unfold immediately and the actual computation is carried out during the folding process, from the base case up. This has the effect that for every colour $q < r$ a recursion on n is started, each of them calculating a list of $n+1$ indices of the corresponding colour. The program performs a "step forwards" only when SE receives some index m of colour q; in this case IS is invoked, asking for the index after $m+1$. On the other hand, an incorrect index for a colour q triggers a "step backwards" and the same index is used as a candidate for the higher colour $q+1$. Since the sequence is finitely coloured, eventually the index will be valid for some colour less than r and the search will continue. The process ends when some list reaches length n, and it is returned as the final result. However, there is one important pitfall: the program IS, called after each "step forwards", restarts $[\![L]\!]^\circ$ from the base case. This invokes fresh recursions for all colours less than q, while the partially accumulated lists for these colours are lost. As a result, $[\![\mathcal{P}]\!]^\circ$ need not necessarily find the first n occurrences of constant colour; it returns a list of the smallest possible indices of a colour q, *between which no colour larger than q appears.*

3.3.3 Extraction via the Dialectica interpretation

In addition to the already used notation $g\lceil n := \lambda k\, g(n \sqcup k)$, we will also employ $g\rceil n := \lambda k\,(n \sqcup gk)$. The substitution $[\bot := \mathrm{F}]$ is implicitly applied to all considered proofs. The extracted programs $[\![L]\!]^+$ and $[\![M]\!]^+$ are defined in Tables 3.3 and 3.4, respectively.

First, let us consider the program $[\![L]\!]^+$. A central role in the program is being played by sequence-extending functions of type $\mathsf{N} \Rightarrow \mathsf{N}$, which when given an index n attempt to provide an index $m \geq n$ at which some given colour q occurs. Such functions are x_w, $[\![L_2]\!]^+$, and $\lambda n_1\, [\![L_3]\!]^-_{v_1}$. When such a function is paired with a colour, like in $[\![K_2]\!]^-_{u_2}$, we obtain the expected computational content of the Infinite Pigeonhole Principle: a way to construct an infinite monochromatic sequence. However, in order to obtain this pair, we need to provide a more complicated parameter: a *challenging function* of type $\mathsf{N} \Rightarrow (\mathsf{N} \Rightarrow \mathsf{N}) \Rightarrow \mathsf{N}$. Such function receives as parameters an infinite monochromatic sequence and attempts to find an index n' at which the sequence-extending function fails. Examples of such functions are x_{u_2} and $[\![K_2]\!]^+$. Finally, the program $[\![L]\!]^+$ returns a pair, the first component of which is a challenge for u_2, i.e., a candidate for an infinite monochromatic sequence, while the second one is a challenge for the assumption that f contains only colours smaller than r, i.e. a single index at which the colour is $\geq r$.

The program $[\![L]\!]^+$ functions by performing backtracking on the statement "the colour r appears infinitely often". This is achieved by the case distinction in $[\![L_4]\!]^-_{u_2}$. The recursively returned challenge $x_p(f\lceil n_1)\lrcorner[\![K_2]\!]^+$ for the assumption $\forall n\,(\neg\neg fn < r)$ is used to find an index at which the colour is not less than r. Since we also assume that the largest colour is r, then a correct counterexample would provide an index of the colour r. Thus it is used to construct the sequence-extending function $\lambda n_1\, [\![L_3]\!]^-_{v_1}$. Then the function x_{u_2} is invoked on r and the constructed sequence-extending function to compute a challenge $n_1\xi$ for that function. The case distinction in $[\![L_4]\!]^-_{u_2}$ determines whether this is a valid counterexample. In case it is not, then an infinite monochromatic sequence of colour r has been found, for which x_{u_2} cannot build a counterexample. Otherwise, we take the infinite monochromatic sequence, which we have from the recursive call. However, the sequence-extending function is modified to always return indices after the computed counterexample $n_1\xi$, since the sequence-extending function for r could not extend the sequence past this index.

Now let us turn to the program $[\![M]\!]^+$. There are a number of challenges being computed and we will review each of them. $[\![M_4]\!]^+$ describes a pair of indices, which challenge the formulas $\mathrm{Decr}(l,n)$ and $\mathrm{Col}(q,l,n)$ respectively. The computation of these counterexamples is recursive and is effectively a linear search through a given list l. Examples of such challenging functions are $[\![M_5]\!]^+$, $[\![M_7]\!]^+$, and $x_{\mathsf{S}n}$. On the other hand, the role of the program $[\![M_2]\!]^+$ is to convert a challenge of $\mathrm{Same}(q,l,n)$ into a challenge of $\mathrm{Col}(l,n)$. The latter is needed for the parameter x_v of the main program $[\![M]\!]^+$, which computes challenges for $\mathrm{Decr}(l,n)$ and $\mathrm{Same}(l,n)$. The monochromatic list of length $n+1$, which is computed recursively by $[\![M_1]\!]^+$, is in fact the challenge for the assumptions u_0 and $u_{\mathsf{S}n}$. In order for the program $[\![L]\!]^+$ to provide a colour q and a sequence-extending function x_w, there needs to be a challenging function to be passed as a parameter. Such a function is computed recursively by $[\![M_8]\!]^+$ by searching for a counterexample for the provided sequence-extending function x_w by testing it on the constructed list. The test is performed by the case distinction in $[\![M_1]\!]^-_w$, which selects the largest index (i.e., earliest appearing in the list) at which the sequence-extending function fails. This challenge achieves the backtracking effect: if the counterexample is valid, then $[\![L]\!]^+$ needs to take a "step backwards" and attempt a different colour; if the counterexample is invalid, then $[\![L]\!]^+$ has constructed a correct sequence-extending function. Finally, the program $[\![M]\!]^+$ provides a pair of outputs: the expected list and a challenge for the assumption that f is finitely coloured, as returned by $[\![L]\!]^+$.

The behaviour of the obtained program is as follows: $[\![L]\!]^+$ constructs a series of candidates for an infinite monochromatic sequence and each of them is being used by $[\![M]\!]^+$ to construct a list of length $n{+}1$. The largest failure index in this list is returned as a counterexample to the currently considered sequence-extending function and is used to construct a sequence-extending function for a higher colour. Since the whole list needs to be constructed before a counterexample is computed, the search for a correct sequence-extending function is quite ineffective. In the next subsection we will show that this property of $[\![M]\!]^+$ is the cause for an exponential average time complexity of the program.

$\mathcal{P}$	$\llbracket\mathcal{P}\rrbracket^+$	$\bullet$	$\llbracket\mathcal{P}\rrbracket^-_\bullet$
K_1	ε	u_1, v_1	$n_1 \sqcup n_2$
$L_1 := \lambda m\, \lambda a\, v_2(n_1 \sqcup m)(L_{\leq} a)$	ε	v_2	$n_1 \sqcup m$
$L_2 := \lambda n_2\, \lambda v_2\, w n_2 L_1$	$\lambda n_2\, (n_1 \sqcup x_w n_2) \equiv x_w \rceil n_1$	w	n_2
$K_2 := \lambda q\, \lambda w\, u_2 q L_2$	$\lambda q\, \lambda x_w\, x_{u_2} q(x_w \rceil n_1)$	u_2	$\langle q, x_w \rceil n_1 \rangle$
$L_3 := p(f\lceil n_1) K_1 K_2$	ε	u_1, v_1	$n_1 \sqcup x_p(f\lceil n_1)\lrcorner \llbracket K_2\rrbracket^+$
		u_2	$\Big\langle x_p(f\lceil n_1)\llcorner \llbracket K_2\rrbracket^+ \llcorner,$ $\big(x_p(f\lceil n_1)\llcorner \llbracket K_2\rrbracket^+ \lrcorner\big)\rceil n_1 \Big\rangle$
		p	$\big\langle f\lceil n_1, \llbracket K_2\rrbracket^+ \big\rangle$
$L_4 := u_2 r(\lambda n_1\, \lambda v_1\, L_3)$	ε	u_1	$\llbracket L_3\rrbracket^-_{u_1}\xi$
		u_2	$\big\langle r, \lambda n_1\, \llbracket L_3\rrbracket^-_{v_1} \big\rangle \overset{u_2}{\bowtie} \llbracket L_3\rrbracket^-_{u_2}\xi$
		p	$\llbracket L_3\rrbracket^-_p \xi$
			$\xi := \big[n_1 := x_{u_2} r(\lambda n_1\, \llbracket L_3\rrbracket^-_{v_1})\big]$
$L_{\mathsf{S}} := \lambda f\, \lambda u_1\, \lambda u_2\, L_4$	$\lambda f\, \lambda x_{u_2}\, \big\langle \llbracket L_4\rrbracket^-_{u_2}, \llbracket L_4\rrbracket^-_{u_1} \big\rangle$	p	$\llbracket L_4\rrbracket^-_p$
L_0	$\lambda f\, \lambda x_{u_2}\, \square^{\mathsf{N}\times(\mathsf{N}\Rightarrow\mathsf{N})\times\mathsf{N}}$		
L	$\lambda r\, \mathcal{R}_{\mathsf{N}} r \llbracket L_0\rrbracket^+ (\lambda r\, \lambda x_p\, \llbracket L_{\mathsf{S}}\rrbracket^+)$		

$$\begin{aligned}
\Big|\forall q\, \tilde{\exists} n\, \forall m\, (n \leq m \to fm = q \to \mathrm{F})\Big|^{x_{u_2}}_{\langle r, \lambda n_1\, \llbracket L_3\rrbracket^-_{v_1}\rangle} &= \Big|\tilde{\exists} n\, \forall m\, (n \leq m \to fm = r \to \mathrm{F})\Big|^{x_{u_2} r}_{\lambda n_1\, \llbracket L_3\rrbracket^-_{v_1}} \\
&= |\forall m\, (n_1\xi \leq m \to fm = r \to \mathrm{F})|^{\varepsilon}_{\llbracket L_3\rrbracket^-_{v_1}\xi} \\
&= n_1\xi \leq m \to fm \neq r, \text{ where } m := \llbracket L_3\rrbracket^-_{v_1}\xi, \\
\text{hence } \llbracket L_4\rrbracket^-_{u_2} &\equiv \mathsf{Cases}(T_{\to}(n_1\xi \leq m)(fm \neq r))\llbracket L_3\rrbracket^-_{u_2}\xi\, \big\langle r, \lambda n_1\, \llbracket L_3\rrbracket^-_{v_1} \big\rangle .
\end{aligned}$$

Table 3.3: Extracted program from the Infinite Pigeonhole Principle by the Dialectica interpretation

$\mathcal{P}$	$\llbracket\mathcal{P}\rrbracket^+$	$\bullet$	$\llbracket\mathcal{P}\rrbracket^-_\bullet$
$M_{\mathsf{S}}^{(i)}$	ε	v_i	$\mathcal{R}_{\mathsf{N}} k\, \Box^{\mathsf{N}}\, (\lambda k\, \lambda p\, p \overset{v_i}{\bowtie} k)$
$M_3 := \lambda m\, \lambda w_1\, \lambda w_2\, u_{\mathsf{S}n}(m :: x :: l) v_0 M'_{\mathsf{S}} M''_{\mathsf{S}}$	ε	$u_{\mathsf{S}n}$	$m :: x :: l$
		v_1	$\llbracket M'_{\mathsf{S}}\rrbracket^-_{v_1}\, [k := x_{\mathsf{S}n}(m :: x :: l)_\llcorner]$
		v_2	$\llbracket M''_{\mathsf{S}}\rrbracket^-_{v_2}\, [k := x_{\mathsf{S}n}(m :: x :: l)_\lrcorner]$
$M_4 := \lambda v_0\, \lambda v_1\, \lambda v_2\, w(\mathsf{S}x) M_3$	$\left\langle \llbracket M_3\rrbracket^-_{v_1}, \llbracket M_3\rrbracket^-_{v_2} \right\rangle [m := x_w(\mathsf{S}x)]$	$u_{\mathsf{S}n}$	$x_w(\mathsf{S}x) :: x :: l$
		w	$\mathsf{S}x$
$M_5 := \lambda l\, \mathsf{Ind}\, l\, \mathsf{efq}(\lambda x\, \lambda l\, \lambda p'\, M_4)$	$\lambda l\, \mathcal{R}_{\mathsf{L(N)}}\, l\, \Box\, (\lambda x\, \lambda l\, \lambda x_{p'}\, \llbracket M_4\rrbracket^+)$	$u_{\mathsf{S}n}$	$\mathcal{R}_{\mathsf{L(N)}}\, l\, \Box\, (\lambda x\, \lambda l\, \lambda x_{p'}\, \llbracket M_4\rrbracket^-_{u_{\mathsf{S}n}})$
		w	$\mathcal{R}_{\mathsf{L(N)}}\, l\, \Box\, (\lambda x\, \lambda l\, \lambda x_{p'}\, \mathsf{S}x)$
$M_{\mathsf{S}} := \lambda u_{\mathsf{S}n}\, p M_5$	$\lambda x_{\mathsf{S}n}\, \llbracket M_5\rrbracket^-_{u_{\mathsf{S}n}} \left[l := x_p \llbracket M_5\rrbracket^+\right]$	w	$\llbracket M_5\rrbracket^-_w \left[l := x_p \llbracket M_5\rrbracket^+\right]$
		p	$\llbracket M_5\rrbracket^+$
$M_6 := \lambda m\, \lambda w_1\, \lambda w_2\, u_0(m{:}) \mathsf{AxT}(\lambda k\, \mathsf{efq})(\lambda k\, w_2)$	ε	u_0	$m{:}$
$M_0 := \lambda u_0\, w 0 M_6$	$\lambda x_0\, x_w 0{:}$	w	0
$M_1 := \mathsf{Ind}\, n\, M_0\, (\lambda n\, \lambda p\, M_{\mathsf{S}})$	$\mathcal{R}_{\mathsf{N}}\, n\, \llbracket M_0\rrbracket^+\, (\lambda n\, \lambda x_p\, \llbracket M_{\mathsf{S}}\rrbracket^+)$	w	$\mathcal{R}\, n\, 0\, (\lambda n\, \lambda p\, p \overset{w}{\bowtie} \llbracket M_{\mathsf{S}}\rrbracket^-_w)$
M_2	$\lambda q\, \lambda k\, k \overset{u}{\bowtie} \mathsf{S}k$		
$M_7 := \lambda l\, \lambda v_0\, \lambda v_1\, \lambda v_2\, v l v_0 v_1 (M_2 q v_2)$	$\lambda l\, \left\langle x_v l_\llcorner, \llbracket M_2\rrbracket^+ q(x_v l_\lrcorner)\right\rangle$	v	l
$M_8 := \lambda q\, \lambda w\, M_1 M_7$	$\lambda q\, \lambda x_w\, \llbracket M_1\rrbracket^-_w$	v	$\llbracket M_1\rrbracket^+ \llbracket M_7\rrbracket^+$
$M\ := \lambda r\, \lambda f\, \lambda z\, \lambda n\, \lambda v\, L r f z M_8$	$\lambda r\, \lambda f\, \left\langle \lambda n\, \lambda x_v\, \llbracket M_8\rrbracket^-_v \left[q, x_w := \llbracket L\rrbracket^+ r f_\llcorner \llbracket M_8\rrbracket^+\right], \lambda n\, \lambda x_v\, \llbracket L\rrbracket^+ r f_\lrcorner \llbracket M_8\rrbracket^+ \right\rangle$		

$$\llbracket M'_{\mathsf{S}}\rrbracket^-_{v_1} \equiv \mathcal{R}\, k\, 0\, (\lambda k\, \lambda p\, \mathsf{Cases}(T_\to (p < n)(l'_{\mathsf{S}p} < l'_p))kp),\ \text{where } l' := x :: l$$

$$\llbracket M''_{\mathsf{S}}\rrbracket^-_{v_2} \equiv \mathcal{R}\, k\, 0\, (\lambda k\, \lambda p\, \mathsf{Cases}(T_\to (p < \mathsf{S}n)(f l'_p = q))kp),\ \text{where } l' := x :: l$$

$$\llbracket M_2\rrbracket^+ \equiv \lambda q\, \lambda k\, \mathsf{Cases}(T_\to (p < \mathsf{S}n)(f l_k = q))(\mathsf{S}k)k,$$

$$\llbracket M_1\rrbracket^-_w \equiv \mathcal{R}\, n\, 0\, \big(\lambda n\, \lambda x_p\, \mathsf{Cases}(T_\to (n \le m)(f m = q)) \llbracket M_{\mathsf{S}}\rrbracket^-_w n\big),\ \text{where } n := x_p \llbracket M_5\rrbracket^+, m := x_w n.$$

Table 3.4: Extracted program from the Unbounded Pigeonhole Principle by the Dialectica interpretation

3.3.4 Comparison

In the present subsection we will compare the programs $[\![\mathcal{P}]\!]^{\circ}$ and $[\![M]\!]^{+}$ in terms of their readability, time complexity and semantics.

The first obvious difference between the two programs is that $[\![M]\!]^{+}$ program is visibly longer and more complicated than its counterpart. One reason for this is the use of substitutions, which noticeably increase the size of the extracted terms. The Dialectica case distinctions also contribute to this problem, as they mention the first candidate counterexample twice: when checked for validity and when returned as a result. However, there is a different source of complexity: $[\![M]\!]^{+}$ also requires an additional parameter x_v and returns an additional result when compared to $[\![\mathcal{P}]\!]^{\circ}$. The phenomenon where the Dialectica extracted programs compute more information than their refined A-translation equivalents was already shown in Section 3.2. This is not necessarily a drawback, especially if it does not increase the asymptotic complexity of the program. The additional parameter x_v seems more concerning, as it would mean that $[\![M]\!]^{+}$ would require more input than $[\![\mathcal{P}]\!]^{\circ}$ in order to compute the answer. However, a careful tracing of the parameter x_v throughout the program reveals that this parameter is never applied and evaluated. Indeed, x_v is used by $[\![M_7]\!]^{+}$, which instantiates the parameters x_0 and $x_{\mathsf{S}n}$ in the recursive program $[\![M_1]\!]^{+}$. During the recursive call $x_{\mathsf{S}n}$ is modified to $[\![M_5]\!]^{+}$, but remains a λ-abstraction. Finally, in the base case of the recursion x_0 is not used at all, which effectively discards the accumulated and complicated functional parameter without executing it even once. Therefore, this parameter only obstructs the readability of $[\![M]\!]^{+}$ without affecting its semantics.

A more careful look shows that the redundant parameters reflect the negative computational content of the formulas $\mathrm{Decr}(l,n)$, $\mathrm{Same}(l,n)$ and $\mathrm{Col}(q,l,n)$ and is generated by the quantifiers $\forall k$. In fact, all these quantifiers are bounded by n, so the three statements are decidable and can be replaced by atomic formulas, effectively removing their computational content. If we modify the proof accordingly, we would extract a program in which the redundant parameter and all terms involving it will be omitted.

In order to estimate time complexity, we need to fix a reduction strategy for terms. We prefer to use a lazy evaluation strategy over a strict evaluation strategy. One reason for this choice is the way terms of the form $\mathcal{R}\,n\,s\,(\lambda n\,\lambda p\,t)$ are evaluated in case $p \notin \mathsf{FV}(t)$. With a strict evaluation strategy n recursion steps will be always performed, but with a lazy strategy we would have only one step, so the recursion will effectively act as a case distinction. If we insisted on strict evaluation strategy, we would need to have a case distinction axiom scheme of the form $\forall n\,(A(0) \to \forall n\,A(\mathsf{S}n) \to A(n))$ and a corresponding conditional computational construct.

There is another redundancy in the extraction of $[\![L]\!]^{+}$: the term $[\![L_4]\!]^{+}_{p}$, which is the negative content of the induction hypothesis, never appears in the final program. The reason is that there is no open assumption in the inductive proof $[\![L]\!]^+$, which would require a recursively defined counterexample. This phenomenon was noticed also by Hernest and Oliva [HO08], where they suggest a finer variant of induction, in which only assumptions which require no challenges are used.

In order to calculate the asymptotic worst time complexity of the programs, we will estimate the number of reductions of expressions involving the recursion operators as a numeric function of the input parameters r and n. The reason is that the recursive reductions are the only ones which directly depend on numeric input parameters, and the total number of reductions in a given execution of the program would be of the same order. We will assume that the functional parameters f and x_v have constant complexity. We will also assume that the operations max, $<$ and $=$ on natural numbers are basic and will not include their evaluation in the total count, since in practice such arithmetic operations are usually implemented by the underlying hardware. In both programs we will give an upper bound on the number of recursion expansions for a given number of colours r, which we will denote as $p(r)$.

For the average time complexity calculation we will estimate the number of steps performed by the programs on uniformly distributed random coloured sequences f. The number of recursive reductions is not a good approximation anymore, because depending on the input sequence some recursions can terminate earlier. We will thus base our estimation on the operational semantics of the programs, which were analysed in Subsections 3.3.2 and 3.3.3.

Worst time complexity of $[\![\mathcal{P}]\!]^\circ$. Applying $[\![\mathcal{P}]\!]^\circ$ to $r+1$ colours would invoke the parameter IMS with the colour r, which will start a recursion on n. In every step of this recursion $[\![M_{\mathsf{S}}]\!]^\circ$ would be evaluated, leading to a single reduction of $\mathcal{R}_{\mathsf{L(N)}}$ and applying the functional parameter IS to a sufficient number of arguments, leading to a step of the main recursion. Since the argument $[\![K_2]\!]^\circ$ has essentially the same behaviour as IMS, we can conclude that

$$p(0) = 1, \qquad\qquad p(r+1) \leq (p(r)+1)\cdot n,$$

which implies that the worst time complexity of $[\![\mathcal{P}]\!]^\circ$ is $O(n^r)$.

Worst time complexity of $[\![M]\!]^+$. As already noted above, the programs $[\![M_5]\!]^+$ and $[\![M_7]\!]^+$ are never reduced, so we can omit them from the analysis. Applying $[\![M]\!]^+$ to a sufficient number of arguments would induce lead to a pair of computations. The right component invokes $[\![L]\!]^+$ directly and in the left component every use of q or x_w would invoke $[\![L]\!]^+$ because of the substitution η. In $[\![M_4]\!]^-_{u_{\mathsf{S}n}}$ we can see that x_w is invoked once for every recursive step of $[\![M_1]\!]^+$. Thus we obtain $n+1$ invocations of $[\![L]\!]^+$ from $[\![M]\!]^+$ for any given number n. Now let consider $[\![L]\!]^+$ being applied to $r+1$ colours and $[\![M_8]\!]^+$, which corresponds to the parameter x_{u_2}. Let us denote by $\sharp t$ the number of invocations of the function x_p in the term t. Because of ξ, every reference to n_1 is in fact an application of x_{u_2}, i.e., $[\![M_8]\!]^+$, which invokes a recursive process on n, such which evaluates x_w on every step a total of three times: twice by the case distinction and one more time in $[\![M_4]\!]^-_{uSn}$. By analysing the programs we obtain the following inequalities:

$$\begin{aligned}
&\sharp[\![L_\mathsf{S}]\!]^+ &&= \sharp[\![L_4]\!]^-_{u_1} + \sharp[\![L_4]\!]^-_{u_2},\\
&\sharp[\![L_4]\!]^-_{u_1} &&= 1,\\
&\sharp[\![L_4]\!]^-_{u_2} &&\le 3(\sharp[\![L_4]\!]^-_{u_1}) + \sharp[\![L_3]\!]^-_{u_2}\xi,\\
&\sharp[\![L_3]\!]^-_{u_2}\xi &&\le 3n+1,\\
\text{hence}\qquad &\sharp[\![L_\mathsf{S}]\!]^+ &&\le 3n+4.
\end{aligned}$$

Additionally, since x_p is invoked to $f\lceil n_1$, every further reference to f in the recursion will result in a calculation of n_1, which is equivalent to $3n$ recursive calls. f is used once on each recursive step in the case distinction $[\![L_4]\!]^-_{u_2}$. Therefore, the number of recursive reductions for a given value of r can be estimated as:

$$p(0) = n+1, \qquad\qquad p(r+1) \le (3nr+3n+4)p(r)+1,$$

hence the worst time complexity of $[\![M]\!]^+$ is $O(r!(3n)^r)$. Note that when viewed as a function on n with r fixed, this is the same as $O(n^r)$. However, if n is kept constant and r varies, $[\![M]\!]^+$ can be seen as having strictly worse time complexity than $[\![\mathcal{P}]\!]^\circ$.

Average time complexity of $[\![\mathcal{P}]\!]^\circ$. Due to the relation between A-translation and CPS, the program $[\![\mathcal{P}]\!]^\circ$ is tail-recursive. The actual calculation of the resulting list happens during the folding process of involved recursions, since the continuations FC and ML are applied to concrete values only in the base cases. In Section 3.3.2 we discussed that when $[\![\mathcal{P}]\!]^\circ$ is executed for given r and n, there are r recursive processes unfolded, one for each colour. Every one of them is a recursion on n, which attempts to construct a list of indices of the respective colour. Moreover, if one of the processes for colour q fails, it is stopped and provides a possible index for the higher colour $q+1$. On the other hand, in order to calculate a next index for colour $q+1$, fresh recursion processes are started for all colours up to q and if all of them fail at a certain index i, then the colour fi cannot be $\le q$, so it is used as a candidate for a next index of $q+1$. Hence, a list of indices of a colour q can be returned only if between the lowest and the highest of these indices there are only colours $\le q$. Therefore, the program $[\![\mathcal{P}]\!]^\circ$ exhibits an asymmetric behaviour by returning only lists of the above kind, which we will refer to as *undisturbed*.

The number of steps executed by $[\![\mathcal{P}]\!]^\circ$ on a given sequence closely follows the largest index in the returned list, because on every recursive call of $[\![M]\!]^\circ$, the parameter IS is called with the successor of the last computed index. Also, the "$\lceil$" guarantees that all following indices will be greater or equal. Having this in mind, we can show that we clearly reach the exponential upper bound. Indeed, consider the following family of initial segments:

$$l^{n,0} := \mathsf{nil} \qquad l^{n,q+1} := \underbrace{l^{n,q}, q, l^{n,q}, q, \dots,}_{n-1} l^{n,q}$$

It is easy to see that $|l^{n,r}| = n^r - 1$. The lists $l^{n,r}$ have the property that they contain no undisturbed subsequence of length n of any color. Note also that appending even a single element to $l^{n,q}$ will break this property, thus such lists are also maximal. Therefore, when run on any infinite sequence starting with $l^{n,r}$, the program $[\![\mathcal{P}]\!]^\circ$ executes n^r recursion steps and the last returned index is $n^r - 1$.

However, the behaviour of $[\![\mathcal{P}]\!]^\circ$ on a random uniformly distributed sequence is much better. Since all lists of indices of the largest colour $r-1$ are trivially undisturbed, such a list is the most probable result. Any colour would appear with a rate $1/r$, i.e., every r indices, hence an undisturbed list of colour $r-1$ and length n would be found on average after checking nr indices. Undisturbed lists of lower colour would be found sooner; in general, the average highest index of an undisturbed list of some colour q is $\leq n(q+1)$. Considering every new index can cost on average $r/2$ recursion folds or unfolds, hence the average time complexity of $[\![\mathcal{P}]\!]^\circ$ can be estimated at $O(nr^2)$.

Remark 3.7. In [RT09], we stated average time complexity of $O(nr)$ under the assumption that considering every index takes a constant amount of time. However, it seems more realistic to assert that indices of higher colour are processed slower, since they require more term reductions. Nevertheless, the program undoubtedly behaves linearly on n as soon as r is fixed.

Average time complexity of $[\![M]\!]^+$. As noted earlier, $[\![M]\!]^+$ starts r recursive processes on n, similarly to $[\![\mathcal{P}]\!]^\circ$. One difference is that some of the processes are executed more than once, because of repeated subterms in the extracted program, and this leads to worse performance for a fixed n and varying r. As is the case with $[\![\mathcal{P}]\!]^\circ$, indices that fail for a given colour q are candidates for the next colour $q+1$. However, the major difference is in the way this "failure index" is calculated. In $[\![K_1]\!]^\circ$, if a failure index for q is found it is immediately passed to the continuation SE, which is building the list for colour $q+1$. However, $[\![M_8]\!]^+$ finds the failure index by a recursive search on n using the provided sequence-extending function x_w. In other words, it constructs the whole list with a fixed function in order to find a failure index. Moreover, in every step there is a case distinction, which leads to a call to x_w and triggers a recursive call for the lower colour. Therefore, $[\![M]\!]^+$ does not perform much better in the average case than in the worst case and hence its average time complexity is still as before $O(r!(3n)^r)$.

A reasonable question is how can the average time complexity of $[\![M]\!]^+$ can be improved at least to the polynomial complexity of $[\![\mathcal{P}]\!]^\circ$. From the comparison it is clear that the culprit is in the inefficient computation of the failure index. In fact, the optimisation mentioned in Subsection 3.2.3 is a solution: we can raise a flag as soon as we have found the first failure index and we will not need to redundantly compute the whole list. With this modification, $[\![\mathcal{P}]\!]^\circ$ would actually produce the same results as $[\![M]\!]^+$ and will achieve lower complexity. In the next chapters, we will demonstrate how this optimisation can be defined generally.

For the sake of comparison, it is interesting to consider a straightforward accumulative algorithm, which computes a witness for a variant[1] of the Unbounded Pigeonhole Principle:

$$\begin{aligned}
L_0 &:= \mathcal{R}\, r\, \mathsf{nil}\, (\lambda k\, \lambda p\ \langle 0, \mathsf{nil}\rangle :: p),\\
\mathsf{ST} &:= \lambda L\, \mathcal{R} L \mathsf{nil}(\lambda x\, \lambda l\, \lambda p\, \lambda i\, \lambda y\, \mathsf{Cases}(i = 0)(y :: l)(p(i-1)y)),\\
P_0 &:= \lambda L\, 0 :: L_{\ulcorner f 0 \lrcorner},\\
P_{\mathsf{S}} &:= \lambda k\, \lambda p\, \lambda g\, \textbf{let }\ q := f(\mathsf{S}k)\ \textbf{ in}\\
&\qquad\qquad\quad \textbf{let }\ m, l := L_q\ \textbf{ in}\\
&\qquad\qquad\quad \textbf{let }\ l' := \mathsf{S}k :: l\ \textbf{ in}\\
&\qquad\qquad\quad \textbf{let }\ L' := \mathsf{ST} L q\, \langle \mathsf{S}m, l'\rangle\ \textbf{ in}\\
&\qquad\qquad\quad \mathsf{Cases}(m = n) l (pL'),\\
P &:= \lambda r\, \lambda f\, \lambda n\, \mathcal{R}(nr + r + 1) P_0 P_{\mathsf{S}} L_0.
\end{aligned}$$

The program P uses the fact that by the finite Pigeonhole Principle $n+1$ occurrences of the same colour must appear within the first $(n+1)r+1$ indices of the sequence. The list $L : \mathsf{L}(\mathsf{N} \times \mathsf{L}(\mathsf{N}))$ accumulates a list of indices of each colour l, together with its length m. The indices are considered in decreasing order and for every index k we put it in front of the list of colour fk. We check if the resulting list l' has reached length $n+1$ and if so, we return it; otherwise we store l' and its length in the accumulator L using the program ST.

In the worst case P would perform $nr + r + 1$ recursive steps and in each of these steps one of the lists would be examined. Taking into account that accessing a list of index q requires at most r steps, the worst time complexity of P is $O(nr^2)$. Since for a uniformly distributed random sequence $O(nr)$ recursive steps would still be

[1]For technical reasons the obtained list of indices is increasing rather than decreasing.

required, the average time complexity is again $O(nr^2)$.

It is important to note that the average time complexity of $[\![\mathcal{P}]\!]^\circ$ is the same as the worst (and average) time complexity of P. This is quite surprising, as the asymmetric backtracking algorithm resulting from the use of classical logic is clearly more inefficient than the direct symmetric algorithm P. Still, it turns out that the program $[\![\mathcal{P}]\!]^\circ$ is as efficient as P in the average case. From a practical point of view, the bad worst case complexity may be a reasonable price to pay for obtaining a correct program from an indirect proof.

3.4 Comparative analysis of the extracted programs

We can make a number observations about the case studies presented in this chapter. First of all, the obtained extracted functional programs are of high type degree, which depends on the complexity of the involved formulas. Even if the conclusion formula is simple, as is the case of the Unbounded Pigeonhole Principle, the type degree of the extracted term depends on the complexity of the Infinite Pigeonhole Principle, which is used as a lemma. Since non-trivial use of classical logic involves formulas of complexity higher than Π^0_2, we can expect that programs extracted from non-constructive proofs are generally of higher type and thus harder to comprehend than programs extracted from direct constructive proofs.

Another notable feature of the obtained programs is the use of backtracking. Although obscured by the operational semantics of the functional programs, with both considered methods we can trace the use of classical logic in the proof to some form of branching, which guides the computation process in one direction or another. Moreover, these directions are usually opposite: either the exploration continues down the computation tree (a “step forwards”), or some partial result is returned to a previous branching point, which can now continue in a different direction (a “step backwards”). Even in the Integer Root example we can view the linear search as a very special case of backtracking, where a “step forwards” continues the search, while a “step backwards” outputs the result.

In the Infinite Pigeonhole Principle case study there is a clear modular structure of the proof, which one of the simplest non-trivial uses of classical logic. We set to prove a Π^0_2 statement as a corollary of a more complicated principle, which is only valid classically. Nevertheless, the proof of the corollary, except for the proof of the classical lemma, is essentially constructive and direct. Naturally, in a more advanced case one could have several classical principles intertwined with essentially constructive arguments. However, generally we could find instances of the interaction between a classical case distinction and a constructive use of it as in the Infinite Pigeonhole Principle case study.

Both considered methods reflect the modular structure of the proof in a certain sense. The common feature of both extracted programs is that the part of the extracted program corresponding to the non-constructive proof L is a backtracking scheme that defines under what condition a "step forwards" or "step backwards" occurs. However, the computation performed on a "step forwards", as well as the value of the branching condition during the computation process depend on the corollary proof M. In a certain sense, the programs $[\![L]\!]^\circ$ and $[\![L]\!]^+$ can be viewed as an "engine", which is being guided by a "driver", whose role is played by the programs $[\![\mathcal{P}]\!]^\circ$ and $[\![M]\!]^+$, respectively, in order to produce concrete results. The programs extracted with the two considered methods are qualitatively different and their specifics can be seen exactly in the interaction between the two aforementioned components of the extracted programs, namely how the "driver" can interact with the "engine" and how the "engine" reacts to the "driver's" input. In the following, we will attempt to identify such general features of the two methods by observing their behaviour on the examples in the present chapter.

3.4.1 Backtracking via refined A-translation

The refined A-translation transforms a proof by substituting $\perp$ with the formula claiming constructive existence of the final witness, in the extracted program the computational type of $\perp$ can be interpreted as the type of this final witness, i.e., the *result type*. Thus, the translated version of the same non-constructive lemma in the subproof results into programs, which have the same term structure, but are annotated with different result types depending on the conclusion formula. Therefore, we can view such "engines" as *polymorphic* or *generic*, specific instances of which are being used by the "driver". The result type specifies one of the forms of interaction between the two components of the extracted program, which allows the "driver" to inject the type of the required witness into the generic "engine".

Programs extracted via refined A-translation are tail-recursive and adhere to the continuation passing style. The name "continuation" traditionally refers to a function that returns the final result depending on certain parameters and thus correspond to negative formulas in the proof. For example, the negative formulation of weak existence as $\tilde{\exists} x\, A := \neg \forall x\, \neg A$ can be viewed as a function, which takes a continuation, corresponding to the false assumption $\forall x\, \neg A$. Continuations are the mechanism, which is used for interaction between the "engine" and the "driver" in programs obtained via refined A-translation. The "driver" provides appropriate continuations, specifying how the "engine" should continue once it reaches a certain state. On the other hand, when the "engine" invokes a continuation with a certain computed result, it essentially provides the "driver" with feedback, which can be used for deciding how to continue, in particular it can be collected as a part of the final result. An example of such interaction is the continuation IMS from Subsection 3.3.2. Whenever IMS is being invoked with a colour q and a continuation IS, the "engine" signals that it attempts to construct a sequence of colour q as specified by the continuation IS. The "driver" responds by invoking a recursive process on n which attempts to collect all found indices of colour q in a list.

Generally speaking, in refined A-translation the backtracking is implemented by nesting two (or more) different calls $f\vec{s}$ and $f\vec{t}$ of the same continuation f within one another. The outer call corresponds to a "step forwards", and the inner call appears inside a continuation parameter passed to the outer call, as follows: $f \ldots (\lambda\vec{x}\, f\vec{t}) \ldots$. Thus a "step backwards" is performed by calling the provided continuation parameter, which corresponds to immediate transfer of the program flow to the alternative branch of the computation. This is only possible, because the result type can appear in the "engine" program, hence it can directly pass the currently computed result to a continuation defined by the "driver". In Subsection 3.3.2 this interaction is exhibited by using a failed index for colour q as a candidate for colour $q+1$, while in Subsection 3.2.2 this corresponds to aborting the search immediately once an integer root is found.

3.4.2 Backtracking via the Dialectica interpretation

The Dialectica interpretation allows for extraction of concrete programs from a non-constructive proof of a formula of an arbitrary complexity. Unlike the case with refined A-translation, here both the "engine" and the "driver" are terms of closed type and an "engine" can be reused with different "drivers" without any instantiation. Thus one can argue that the Dialectica interpretation is in a sense more modular than the refined A-translation.

It is interesting to compare the branching points in programs generated by the two different methods. With refined A-translation the case distinctions generally emerge from a case analysis on a negated formula in the original MA^ω proof in or from the translation regarding definite and goal formulas. On the other hand, in the Dialectica interpretation repeated use of the same assumptions (i.e., contractions) are the clear source of case distinction. This has an interesting effect in the case of the Infinite Pigeonhole Principle case study: the case distinctions in $[\![\mathcal{P}]\!]^\circ$ are generated from the *antecedent* of the lemma (i.e., the sequence is finitely coloured), while in $[\![M]\!]^+$ the case distinction comes from the Dialectica interpretation of the *consequent* of the lemma.

The Dialectica extracted program can refer only to the types in its formula and hence values and functions constructed from these types can be the only means of interaction between an "engine" and a "driver". The dual meaning of Dialectica programs as computation of realisers and challenges enables the communication between two components of a program. Thus, in order for the "engine" to produce a realiser, it requires a challenging function from the "driver", attempting to produce counterexamples to candidates for realisers. Such counterexample candidates in fact guide the backtracking process, because they can be plugged into the decidable Dialectica translation and their validity can be directly verified. An invalid counterexample informs the "engine" that it is moving in the right direction and it makes a "step forwards". On the contrary, a valid counterexample from the "driver" is a sign that the engine needs to take a "step backwards" and attempt another branch. An example of such interaction can be seen in the Dialectica interpretation of induction, where challenges are computed by recursion, where on each step there is a choice between a new challenge (a "step forwards") and a recursively computed challenge (a "step backwards"). An instance of this example is the program $[\![L_4]\!]^-_{u_2}$ in Table 3.3, which can be interpreted as follows: in case the "driver" managed to show that the candidate sequence-extending function fails for r, then the "engine" continues the recursive search for a sequence-extending function for a lower colour; otherwise, if the "driver" did not find a suitable counterexample, then the provided sequence-extending function works and is returned as a result.

As already noted for the Integer root example, this mechanism has a major drawback, namely that during the recursive computation of a counterexample candidate, all possibilities are checked, where in fact only the first valid counterexample could be sufficient. If it happens that the case distinction itself invokes recursion in the "engine", as in the computation of $[\![M_1]\!]^-_w$ in Table 3.4, the resulting program can have an exponential slowdown, as shown in Subsection 3.3.3. This problem does not occur in the refined A-translation, because the interaction between the "driver" and the "engine" is much more direct: as soon as some counterexample is found, it is immediately passed to the other party by the appropriate continuation, instead of blindly continuing the search.

3.4.3 Computational inefficiencies of the Dialectica interpretation

All the case studies presented in this chapter confirm that programs extracted via the Dialectica interpretation are outperformed by their counterparts extracted via refined A-translation. A natural questions is can these drawbacks be remedied in a general way, so that the Dialectica extracted programs become at least as good as their alternatives.

We can summarise the inefficiencies in the extracted programs in three major categories:

1. Substitution can lead to subterm repetition and recomputation, producing unnecessarily larger and slower programs,
2. Dialectica programs can require superfluous parameters or produce superfluous results,
3. Recursive counterexample search examines all possibilities, even if redundant.

The following chapters present general solutions to the problems outlined above. Chapter 4 defines a variant of the Dialectica interpretation that allows to establish an almost linear bound on the size of the extracted term by eliminating unnecessary substitutions. Chapter 5 explains how unnecessary parameters and results can be omitted from the extracted terms by employing *uniform annotations*, as introduced by Berger [Ber05] and adapted to Dialectica by Hernest [Her07a, Her07b]. We extend the uniform annotations for the Dialectica interpretation to allow the finest level of computational control. Finally, Chapter 6 identifies the cases, in which redundant recursive calls can appear and demonstrates how they can be avoided through the use of counterexample markings.

CHAPTER

FOUR

QUASI-LINEAR DIALECTICA INTERPRETATION

An easily noticeable defect of programs extracted via the Dialectica interpretation is the repetition of equal subterms. In the Soundness Theorem 2.21 the cause can be easily traced back to an asymmetry in the treatment of elimination rules for the conclusion and for the assumptions. While for positive content an elimination rule corresponds to *application* to a term t, for negative content we *substitute* a challenge variable with a pair of the form $\langle t, y\rangle$. Naturally, the challenge variable could have multiple appearances and this can lead to multiplication of the substituted terms. This fact certainly impacts the length and the readability of the extracted term. However, it can have also consequences for the complexity of the obtained program, because equal subterms may be redundantly evaluated.

In this chapter we will present a syntactic reformulation of the Dialectica interpretation, which allows for extraction of terms in which syntactic repetition of subterms is avoided as much as possible. As a result the size of extracted terms will depend almost linearly on the size of the proof. This optimisation will naturally lead to complexity improvements in certain cases. Most of the results presented in this chapter have been published in [Tri10b].

4.1 Examples of recomputation

Syntactic repetition of the same term does not always imply its reevaluation. For example, let us consider the term $\mathsf{Cases}\,((\mathsf{S}t)^2 > 0)\, s\, t$. Even though t appears twice in the term, the condition will always evaluated to tt, so the alternative branch will never be evaluated. This situation is usually referred to as “dead code”. Another example can be given with the term $(\lambda y\, \lambda z\, \mathsf{S}y)t\, t$. Under a strict evaluation strategy t will be evaluated twice, however, a lazy evaluation strategy would evaluate only the first occurrence of t which is used to instantiate y. However, the subterm t will be reevaluated under any reduction strategy in the term $(\lambda x\,\lambda y\,x+y)t\,t$, even though it has the same normal form as $(\lambda x\,x+x)t$, in which there is no reevaluation.

Naturally, evaluating equal subterms multiple times would result in slower computation as opposed to evaluating the subterm just once and reusing the value. The following example shows that substitution on the extraction level could lead to a program with exponential time complexity, even if the underlying process is polynomial.

Example 4.1. Let us prove the totality of the function $x, y \mapsto 2^x(x+y)$, i.e.,

$$\forall x\,\forall y\,\tilde{\exists} z\,(z = 2^x(x+y)).$$

A simple and essentially constructive proof goes by induction on x, taking $z := y$ for the base case. For the step case our induction hypothesis is $\forall y\,\tilde{\exists} z\,\big(z = 2^x(x+y)\big)$, so we fix y_0 and look for a $z_0 = 2^{x+1}(x+y_0+1)$. We use the induction hypothesis on y_0+1 to find a z and then take $z_0 := z+z$. The formal proof of the statement is given below:

$$\begin{aligned}
M &:= \lambda x\,\mathsf{Ind}_{\mathsf{N}}\,x\,(\lambda y\,\lambda u_0\,u_0\,y\,\mathsf{AxT})\ (\lambda x\,\lambda p\,\lambda y\,\lambda u_{\mathsf{S}x}\,p(\mathsf{S}y)(\lambda z\,\lambda v\,u_{\mathsf{S}x}(z+z)(L_{=}v)))\,,\\
u_i &: \forall z\,(z = 2^i(i+y) \to \bot),\\
v &: z = 2^x(x+\mathsf{S}y),\\
L_{=} &: z = 2^x(x+\mathsf{S}y) \to z+z = 2^{\mathsf{S}x}(\mathsf{S}x+y).
\end{aligned}$$

With refined A-translation we can directly apply Theorem 2.13 to the proof

$$\mathcal{P} := Mxy\,[\bot := \exists z\,(z = 2^x(x+y))]\,\exists^+,$$

and we obtain

$$[\![\mathcal{P}]\!]^\circ \equiv \mathcal{R}_{\mathsf{N}}\,x\,(\lambda y\,\lambda h^{\mathsf{N}\Rightarrow\mathsf{N}}\,hy)\,(\lambda x\,\lambda p\,\lambda y\,\lambda h^{\mathsf{N}\Rightarrow\mathsf{N}}\,p(\mathsf{S}y)(\lambda z\,h(z+z)))y(\lambda z\,z).$$

$[\![\mathcal{P}]\!]^\circ$ acts by accumulating the result in the continuation h. Once the recursion bottom is reached, the folding process starts and performs x additions in order to compute the final result. The time complexity of the process is clearly $O(x)$, regardless whether the evaluation strategy is strict or lazy.

On the other hand, when we use Dialectica interpretation on the proof $M\,[\bot := \mathrm{F}]$, we obtain

$$[\![M]\!]^+ \equiv \lambda x\,\mathcal{R}_{\mathsf{N}}\,x\,(\lambda y\,y)(\lambda x\,\lambda x_p\,\lambda y\,x_p(\mathsf{S}y) + x_p(\mathsf{S}y)).$$

The program $[\![M]\!]^+$ generates a tree-recursive process, which leads to a number of additions which is exponential on x. Thus the time complexity of the program is $O(2^x)$ and this behaviour is not affected by the choice of evaluation strategy. Clearly, the cause for the exponential behaviour is the repeated appearance of the term $x_p(\mathsf{S}y)$. This repetition comes from the interpretation of the subproof L_1L_2 where $L_1 := p(\mathsf{S}y)$ and $L_2 := \lambda z\, \lambda v\, u_{\mathsf{S}x}(z+z)(L_= v)$. Since $[\![L_1]\!]^+ \equiv x_p(\mathsf{S}y)$ and $[\![L_2]\!]^-_{u_{\mathsf{S}x}} \equiv z+z$, by Theorem 2.21 we obtain

$$[\![L_1L_2]\!]^-_{u_{\mathsf{S}x}} \equiv [\![L_2]\!]^-_{u_{\mathsf{S}x}} \left[z := [\![L_1]\!]^+\right] \equiv x_p(\mathsf{S}y) + x_p(\mathsf{S}y).$$

We see that the duplication is caused by the substitution of the challenge variable z which has two occurrences. If instead substitution we would have used a **let**-construction, we would have obtained the program

$$\lambda x\, \mathcal{R}_{\mathsf{N}}\, x\, (\lambda y\, y)(\lambda x\, \lambda x_p\, \lambda y\, \mathbf{let}\ z := x_p(\mathsf{S}y)\ \mathbf{in}\ z+z),$$

which is of linear complexity both under strict and lazy evaluation strategy, where for the latter memoisation of evaluated arguments as in the programming language Haskell is assumed.

Generally, repeated subterms may or may not occur in an extracted program depending on two factors: the proof and the extraction method. Our goal will not be to eliminate subterm repetition altogether, but only these repeated instances which are generated by the Dialectica interpretation. We assume the input proof to be external and we do not try to optimise it, but rather reflect its modular structure as close as possible in the extracted term. In Example 4.1 the repetition of subterms is definitely generated by the interpretation rather than the proof, since in the proof M the only repeated term is the variable z, and the substitution which causes the tree recursion is a part of the extraction method. If, on the other hand, in the proof M we had redundantly used the induction hypothesis twice in the following manner:

$$\dots p(\mathsf{S}y)\big(\lambda z_1\, \lambda v_1\, p(\mathsf{S}y)(\lambda z_2\, \lambda v_2\, u_{\mathsf{S}x}(z_1+z_2)(L'_= v_1 v_2))\big)\dots,$$

then the source of repetition would already be the proof and we should not expect that such repetition is removed by the extraction method. Recomputations also occur in the examples in Sections 3.1 and 3.3; in the latter recomputations lead to an exponential Dialectica program of higher exponent than its refined A-translation counterpart in case n is fixed and r varies.

4.2 Towards avoiding syntactic repetition

When inspecting Theorem 2.21 we can notice that substitution is only needed to construct the negative extracted terms, while for the positive ones substitution is never used. In particular, the Dialectica interpretation treats positive witnesses very similarly to the modified realisability interpretation. Witnesses are built “outwards” by application or abstraction of previous witnessing terms without the necessity to use substitution. In contrast, challenges grow “inwards” by substituting the previous challenge variable by a term containing a new challenge variable. For introduction rules such substitution is mostly harmless since they are of the form $[y := y'\llcorner]$ and $[y := y'\lrcorner]$. Although such substitutions could still lead to a non-constant increase in term size, they could not have a significant impact on asymptotic time complexity. On the other hand, in elimination rules challenge variables are substituted with previously computed witnesses or challenges, which could be arbitrarily complex. Moreover, this situation is not specific to the natural deduction treatment; substitution in the negative content is used to prove the soundness also in [Tro73], for example for the axiom $Q2$, corresponding to $\forall$-elimination.

The problem is even worse for the interpretation of induction. Let us recall the extracted terms for a proof by natural induction $\mathsf{Ind}_{\mathsf{N}}^{n,A}\, n\, M_1^{A[n:=0]}\, (\lambda n\, \lambda u_0^A\, M_2^{A[n:=\mathsf{S}n]})$:

$$[\![\mathcal{P}]\!]^+ := \mathcal{R}_{\mathsf{N}}^{\tau^+(A)}\, n\, [\![M_1]\!]^+(\lambda n\, \lambda x_0\, [\![M_2]\!]^+),$$
$$[\![\mathcal{P}]\!]_i^- := \mathcal{R}_{\mathsf{N}}^{\tau^-(A)\Rightarrow\tau^-(C_i)}\, n\, (\lambda y_A\, [\![M_1]\!]_i^-)\Big(\lambda n\, \lambda p\, \lambda y_A\, (p[\![M_2]\!]_0^- \overset{C_i}{\bowtie} [\![M_2]\!]_i^-)\xi\Big) y_A,$$

where $\xi := \big[x_0 := [\![\mathcal{P}]\!]^+\big]$. Note that the witnessing variable x_0, corresponding to the positive content of the induction hypothesis, is substituted with the recursively computed positive witness $[\![\mathcal{P}]\!]^+$. Therefore, multiple nested recursions on n may be generated for every open assumption variable used in the inductive proof.

One solution to this problem would be to use **let**-constructions of the form **let** $x := s$ **in** t, instead of substitutions $t\,[x := s]$. The two alternatives have the same normal form, but the former is generally shorter and stores explicitly the information that multiple occurrences of the same term are related. This simple trick reduces the repetitions, but unfortunately does not eliminate them completely. The reason lies in the dual nature of the Dialectica interpretation: the same proof is used to generate two kinds of computational content: positive and negative. Let us consider the case of a proof by universal quantifier elimination Mt. Using a **let**-construction we have $[\![Mt]\!]^+ \equiv [\![M]\!]^+ t$ and $[\![Mt]\!]_i^- \equiv$ **let** $y := \langle t, y'\rangle$ **in** $[\![M]\!]^-$. Nevertheless, when the open assumptions are eventually eliminated, we will obtain a program which will contain as subterms $[\![Mt]\!]^+$ and all $[\![Mt]\!]_i^-$ and therefore will still have $n+1$ occurrences of t, if n is the number of open assumptions in the proof M. Therefore, in order to eliminate all repetitions caused by the extraction, we would need to compute both positive and negative computational content simultaneously, so that we can abstract the common subterms with a single **let**-construction. Consequently, the Dialectica computational types needs to be syntactically reformulated so that we are able to combine positive and negative computational content emerging from the same proof.

In the next sections we will expand on this idea, which will result in a size bound on the extracted terms, which is almost linear on the size of the input proof. A similar complexity result was already obtained by Hernest and Kohlenbach in [HK05]. One major presentational difference between the current exposition and the results in [HK05] is the use of a natural deduction system and λ-calculus versus Hilbert-style equational logic with Schönfinkel style combinators Σ and Π. On page 229 the authors comment:

> Smaller terms can be extracted if we use a simplification provided by the definitional equation of Σ. The size of the extracted terms becomes linear in the size of the proof at input. Nevertheless the use of extra Σ's brings an increase in type complexity. This can be avoided by using a more economical representation of the realizing tuples by means of pointers to parts which are shared by all members of a tuple.

While the idea for sharing common subterms across all extracted components is conceptually the same as what is being proposed in this thesis, the authors of [HK05] consider such a representation to be implicit and related to the internal representation of the term. On the other hand, we will display concrete λ-terms, which satisfy a roughly linear bound regardless of their internal representation, while the sharing is being made explicit. The downside of this approach is that because of the technical subtleties in dealing with the package of positive and negative witnesses, we do not obtain a strictly linear bound like in [HK05], but a bound which is "almost" linear for practical applications.

4.3 Definition contexts

We will factor out common subterms by using a *definition context* — a term containing a single occurrence of a *hole* $[]$, which can be instantiated with any other term. In order to define this notion, let us reserve a type variable $\diamond$ and an object variable of this type $[] : \diamond$, which we will call *a hole.*

Definition 4.2 (Definition context)**.** A *definition context* E is a term built by the following rules:

$$E \quad ::= \quad []^{\diamond} \mid (E^{\rho\Rightarrow\sigma} t^{\rho})^{\sigma} \mid (\lambda x^{\rho}\, E^{\sigma})^{\rho\Rightarrow\sigma},$$

where $\diamond$ does not appear in the type in any subterm of t.

A *term context* is an *arbitrary* term with exactly one occurrence of a hole $[]$. Thus any definition context is a term context with certain syntactic restrictions on the position where the hole occurs.

For an arbitrary term context E^ρ and term t^σ, we define the term $E[t]$ (pronounced "t in the context E,') as $E\,[\diamond := \sigma]\,[[] := t]$, where, contrary to our usual convention, the free variables of t are allowed to be bound by abstractions in E.

Proposition 4.3. *Let E_1 and E_2 be definition (term) contexts. Then $E_1[E_2]$ is a definition (term) context.*

Proof. For the case of definition contexts we use induction on the definition of E_1. The case $E_1 \equiv []$ is trivial. Assume $E_1 \equiv E_3 t$. Then, by induction hypothesis $E_3[E_2]$ is a definition context, and hence so is $E_3[E_2]t \equiv E_1[E_2]$. Similarly, for $E_1 \equiv \lambda x^\rho\, E_3$, by induction hypothesis $\lambda x^\rho\, E_3[E_2] \equiv E_1[E_2]$.

The case of term contexts is trivial, since $E_1[E_2]$ will still contain a single occurrence of $[]$. □

Corollary 4.4. *If E is a definition context, x is a variable and t is an arbitrary term, then **let** $x := t$ **in** E is also a definition context.*

Any definition context E has the type $\vec{\rho} \Rightarrow \diamond$ for some list of types $\vec{\rho}$. A key property of the definition contexts is that they correspond to substitutions in a certain way. Indeed, if we have a substitution $\Xi := [\vec{x} := \vec{s}]$, then we can define the context $E :=$ **let** $\vec{x} := \vec{s}$ **in** $[]$ and for every term t we will have $E[t] \overset{r}{=} t\Xi$. The reverse correspondence is given by the following property.

Proposition 4.5 (Context property)**.** *For every definition context $E : \vec{\rho} \Rightarrow \diamond$ and a list of different variables $y_i : \rho_i$ there is a substitution Ξ, such that $(E\vec{y})[t] \overset{r}{=} t\Xi$ for any term t.*

Proof. Induction on the definition of E.

Case $E = []$. Take Ξ to be the identity substitution.

Case $E = E's$. By induction we have Ξ' such that $(E'y_0\vec{y})[t] \overset{r}{=} t\Xi'$ for any t. For $\Xi := \Xi'\,[y_0 := s]$ we obtain the desired property.

Case $E = \lambda x^{\rho_1}\, E'$. By induction we have Ξ' such that $(E'y_2 \dots y_n)[t] \overset{r}{=} t\Xi'$ for any t. Take $\Xi := \Xi'\,[x := y_1]$. Then we will have

$$\big((\lambda x\, E')y_1 y_2 \dots y_n\big)[t] \equiv \big((\lambda x\, E'[t])y_1 y_2 \dots y_n\big) \overset{r}{=} \big(E'[t]y_2 \dots y_n\big)\,[x := y_1] \overset{r}{=} t\Xi.$$

□

The context property implies that under special circumstances we can permute contexts over application.

Corollary 4.6. *For every definition context $E : \vec{\rho} \Rightarrow \diamond$, variables $y_i : \rho_i$ and terms $t^{\sigma \Rightarrow \tau}$ and s^σ, such that $\mathsf{FV}(t) \cap \mathsf{BV}(E) = \emptyset$ we have $E[ts] \overset{r}{=} \lambda\vec{y}\,\big(tE\vec{y}[s]\big)$.*

Proof. Take the substitution Ξ from Proposition 4.5. By the variable condition for t we have $t\Xi \equiv t$. Then we obtain

$$E[ts] \overset{r}{=} \lambda\vec{y}\, E\vec{y}[ts] \overset{r}{=} \lambda\vec{y}\,(ts)\Xi \equiv \lambda\vec{y}\, t(s\Xi) \overset{r}{=} \lambda\vec{y}\, t(E\vec{y}[s]). \qquad \square$$

A definition context will be used to hold all common subterms of the extracted computational content, and the hole will be instantiated with a tuple of context-dependent terms specifically corresponding to positive and negative computational content. In order to keep terms as small as possible, we will delay hole substitution until the last possible moment. The pairing operation $\langle\cdot,\cdot\rangle$ will allow us to bundle together an arbitrary number of differently typed terms. We will introduce the following notation to easily access components of such tuples:

$$\begin{aligned} t^\rho \triangleright 0 &:= t, \qquad \text{if } \rho \text{ is not a product,}\\ t^{\rho\times\sigma} \triangleright 0 &:= t_\llcorner,\\ t^{\rho\times\sigma} \triangleright (i+1) &:= t_\lrcorner \triangleright i. \end{aligned}$$

4.4 Some syntactic notions

We will use the following notions of size of terms, formulas and proof.

Definition 4.7 (Term size)**.** For a term t its size $\lceil t \rceil$ is defined inductively:

- $\lceil C \rceil := 1$ if C is a variable, a constructor or a recursion constant.
- $\lceil \lambda x\, t \rceil := \lceil t_\llcorner \rceil := \lceil t_\lrcorner \rceil := \lceil t \rceil + 1$
- $\lceil st \rceil := \lceil \langle s, t\rangle \rceil := \lceil s \rceil + \lceil t \rceil + 1$

Definition 4.8 (Formula size)**.** For a formula A its size $\lceil A \rceil$ is defined inductively:

- $\lceil \mathrm{at}(t) \rceil := \lceil t \rceil$
- $\lceil A \to B \rceil := \lceil A \rceil + \lceil B \rceil + 1$
- $\lceil \forall x\, A \rceil := \lceil \exists x\, A \rceil := \lceil A \rceil + 1$

Consequently, $\lceil \neg A \rceil = \lceil A \rceil + 2$, $\lceil \tilde{\exists} x A \rceil = \lceil A \rceil + 5$.

Definition 4.9 (Proof size)**.** For a proof M its size $\lceil M \rceil$ is defined inductively:

- $\lceil u^A \rceil := \lceil A \rceil$ if u is an axiom instance or an assumption variable
- $\lceil \lambda u^A\, M \rceil := \lceil M \rceil + \lceil A \rceil$
- $\lceil \lambda x^\rho\, M \rceil := \lceil M \rceil + 1$
- $\lceil MN \rceil := \lceil M \rceil + \lceil N \rceil + 1$
- $\lceil Mt \rceil := \lceil M \rceil + \lceil t \rceil + 1$

Remark 4.10. Note that the definition of size for assumption variables depends on the size of the formula they assume, but the size of object variables is defined to be constant. The reason for this asymmetry is that type annotations for terms can be actually implicit ("Curry style"), since types can be decidably inferred in the considered term system. Conversely, inference is not decidable for formula annotations for proofs in NA^ω. Hence, for practical reasons we do not account for types in the definition of term size.

Definition 4.11 (Maximal sequent length)**.** For a proof M we define its maximal sequent length $\llbracket M \rrbracket$ as

$$\llbracket M \rrbracket := \max_{N \leq M} \lceil \mathsf{FA}(N) \rceil,$$

where $N \leq M$ is the subproof relation.

Extracted tuples will usually be of the form $\left\langle \llbracket M \rrbracket^+, \dots, \llbracket M \rrbracket_u^-, \dots \right\rangle$ for some proof M. We will denote $t \triangleright u$ for u an assumption variable instead of a numerical index, in order to access the respective negative content $\llbracket M \rrbracket_u^-$ from the tuple t in a more convenient manner. In order to unify positive and negative content, the definition of the Dialectica computational types needs to be slightly revised so that we use uncurried function types instead of curried ones. The reason for this convention is practical: while both the partial and the full application of an uncurried function to a variable increase the term size with a constant, full application of a curried function needs a variable for each parameter. We will redefine the computational types of the Dialectica interpretation so that they are normal with respect to the reduction rules

$$\begin{aligned} \rho \Rightarrow \sigma \Rightarrow \tau \quad &\rightsquigarrow \quad \rho \times \sigma \Rightarrow \tau, \\ (\rho \Rightarrow \sigma) \times (\rho \Rightarrow \tau) \quad &\rightsquigarrow \quad \rho \Rightarrow (\sigma \times \tau). \end{aligned} \tag{$\rightsquigarrow$}$$

However, this would mean that during extraction we might need to apply a function of type $\rho \times \sigma \Rightarrow \tau$ to a value of type ρ. We will use the following notation for such a partial application and its dual partial abstraction

$$\begin{aligned} f^{\rho \Rightarrow \tau} \circ t^\rho &:= ft \\ f^{\rho \times \sigma \Rightarrow \tau} \circ t^\rho &:= \lambda x^\sigma\, f\, \langle t, x \rangle\,, \text{ where } x \text{ is a fresh variable,} \\ \lambda^\circ x^\rho\, t^{\sigma \Rightarrow \tau} &:= \lambda y^{\rho \times \sigma}\, \mathbf{let}\ x := y_{\llcorner}\ \mathbf{in}\ t(y_{\lrcorner}). \end{aligned}$$

Finally, we extend the projection operations $\llcorner$ and $\lrcorner$ to functions as follows:

$$f^{\rho \Rightarrow \sigma \times \tau}{}_{\llcorner} := \lambda x^\rho\, f x_{\llcorner}, \qquad f^{\rho \Rightarrow \sigma \times \tau}{}_{\lrcorner} := \lambda x^\rho\, f x_{\lrcorner}.$$

The following property can be easily checked.

Proposition 4.12.

1. $\lceil \mathsf{Cases}\, bst \rceil = \lceil b \rceil + \lceil s \rceil + \lceil t \rceil + 4$
2. $\lceil \textbf{\textit{let}}\ x := s\ \textbf{\textit{in}}\ t \rceil = \lceil (\lambda x\, t) s \rceil = \lceil s \rceil + \lceil t \rceil + 2$
3. $\lceil E[t] \rceil = \lceil E \rceil + \lceil t \rceil - 1$
4. $\lceil t \triangleright i \rceil \leq \lceil t \rceil + i + 1$
5. $\lceil f^{\rho \times \sigma \Rightarrow \tau} \circ t^{\rho} \rceil \leq \lceil f \rceil + \lceil t \rceil + 4$
6. $\lceil \lambda^{\circ} x^{\rho}\, t^{\sigma \Rightarrow \tau} \rceil \leq \lceil t \rceil + 6$
7. $\lceil f^{\rho \Rightarrow \sigma \times \tau} \llcorner \rceil = \lceil f^{\rho \Rightarrow \sigma \times \tau} \lrcorner \rceil = \lceil f \rceil + 4$

4.5 Quasi-linear extraction

In this section we will revise the definitions of positive and negative computational types, as well as the definition of the Dialectica translation. The purely syntactical changes will aid defining a recomputation-free variant of the interpretation. However, unlike Theorem 2.21, we will work in NA^{ω} instead of HA^{ω}. This choice aims to avoid the overcomplicated bureaucratic treatment of pairs of extracted terms arising from the interpretation of the introduction rules for conjunction and strong existence.

Definition 4.13 (Quasi-linear computational types)**.** For a formula $A \in \mathrm{NA}^{\omega}$ we will redefine the positive and negative computational types and denote the new variants as $\sigma^{+}(A)$ and $\sigma^{-}(A)$. We will also denote $\sigma^{*}(A) := \sigma^{-}(A) \Rightarrow \sigma^{+}(A)$. We define:

$$\begin{aligned}
\sigma^{+}(\mathrm{at}(b)) &:= \mathsf{I}, & \sigma^{-}(\mathrm{at}(b)) &:= \mathsf{I},\\
\sigma^{+}(A \to B) &:= \sigma^{+}(B) \times \sigma^{-}(A), & \sigma^{-}(A \to B) &:= \sigma^{*}(A) \times \sigma^{-}(B)\\
\sigma^{+}(\forall x^{\rho} A) &:= \sigma^{+}(A), & \sigma^{-}(\forall x^{\rho} A) &:= \rho \times \sigma^{-}(A),
\end{aligned}$$

The relation with the original definition of computational types can be established up to the reduction relation ($\rightsquigarrow$).

Proposition 4.14. *For every* NA^{ω} *formula* C*,*

$$\tau^{-}(C) \rightsquigarrow^{*} \sigma^{-}(C), \qquad \tau^{+}(C) \rightsquigarrow^{*} \sigma^{*}(C).$$

Proof. Induction on the formula C.
Case $\mathrm{at}(t)$. Trivial.
Case $A \to B$.

$$\begin{aligned}
\tau^{+}(A \to B) &= (\tau^{+}(A) \Rightarrow \tau^{+}(B)) \times (\tau^{+}(A) \Rightarrow \tau^{-}(B) \Rightarrow \tau^{-}(A))\\
&\rightsquigarrow^{*} \big(\sigma^{*}(A) \Rightarrow \sigma^{*}(B)) \times (\sigma^{*}(A) \Rightarrow \sigma^{-}(B) \Rightarrow \sigma^{-}(A)\big)
\end{aligned}$$

$$\begin{aligned}
&\rightsquigarrow^* && \big(\sigma^*(A) \Rightarrow \sigma^-(B) \Rightarrow \sigma^+(B)\big) \times \big(\sigma^*(A) \Rightarrow \sigma^-(B) \Rightarrow \sigma^-(A)\big)\big) \\
&\rightsquigarrow^* && \big(\sigma^*(A) \times \sigma^-(B) \Rightarrow \sigma^+(B)\big) \times \big(\sigma^*(A) \times \sigma^-(B) \Rightarrow \sigma^-(A)\big)\big) \\
&\rightsquigarrow && (\sigma^*(A) \times \sigma^-(B)) \Rightarrow (\sigma^+(B) \times \sigma^-(A)) \\
&= && \sigma^-(A \to B) \Rightarrow \sigma^+(A \to B) \quad = \quad \sigma^*(A \to B), \\
\tau^-(A \to B) \quad &= && \big(\tau^+(A) \times \tau^-(B)\big) \rightsquigarrow^* \big(\sigma^*(A) \times \sigma^-(B)\big) \\
&= && \sigma^-(A \to B).
\end{aligned}$$

Case $\forall x^\rho\, A$.

$$\begin{aligned}
\tau^+(\forall x\, A) \quad &= \quad \rho \Rightarrow \tau^+(A) \quad \rightsquigarrow^* \quad \rho \Rightarrow \sigma^*(A) \\
&= \quad \rho \Rightarrow \sigma^-(A) \Rightarrow \sigma^+(A) \\
&\rightsquigarrow \quad \rho \times \sigma^-(A) \Rightarrow \sigma^+(A) \\
&= \quad \sigma^-(\forall x\, A) \Rightarrow \sigma^+(\forall x\, A) \quad = \quad \sigma^*(\forall x\, A), \\
\tau^-(\forall x\, A) \quad &= \quad \rho \times \tau^-(A) \quad \rightsquigarrow^* \quad \rho \times \sigma^-(A) \quad = \quad \sigma^-(\forall x\, A).
\end{aligned}$$ $\square$

Next, we will define bidirectional term transformations corresponding to the syntactic changes in the computational type.

Definition 4.15 (Quasi-linear transformations)**.** Let C be an arbitrary NA$^\omega$ formula. By simultaneous induction on the formula A, we will define transformations $(\cdot)^{\uparrow\pm}$ and $(\cdot)^{\downarrow\pm}$, which transform terms of the respective computational types, as shown on Figure 4.1.

$$\begin{tikzcd}
\sigma^*(C) \arrow[d, bend left, "+"] & \sigma^-(C) \arrow[d, bend left, "-"] \\
\tau^+(C) \arrow[u, bend left, "+"] & \tau^-(C) \arrow[u, bend left, "-"]
\end{tikzcd}$$

Figure 4.1: Transformations between the Dialectica computational types

For $C = \mathrm{at}(r)$, we define $\varepsilon^{\updownarrow\pm} := \varepsilon$.
For $C = A \to B$, we define

$$\begin{aligned}
t^{\uparrow+} &:= \lambda x^{\sigma^-(C)} \left\langle (t_\llcorner (x_\llcorner)^{\downarrow+})^{\uparrow+}(x_\lrcorner), (t_\lrcorner (x_\llcorner)^{\downarrow+}(x_\lrcorner)^{\downarrow-})^{\uparrow-} \right\rangle, \\
t^{\downarrow+} &:= \left\langle \lambda y^{\tau^+(A)} ((t \circ y^{\uparrow+})_\llcorner)^{\downarrow+}, \lambda y^{\tau^+(A)} ((t \circ y^{\uparrow+})_\lrcorner)^{\downarrow-} \right\rangle, \\
t^{\updownarrow-} &:= \left\langle (t_\llcorner)^{\updownarrow+}, (t_\lrcorner)^{\updownarrow-} \right\rangle.
\end{aligned}$$

For $C = \forall x^\rho A$, we define

$$\begin{aligned} t^{\uparrow +} &:= \lambda y^{\sigma^-(C)}\, (t(y_\llcorner))^{\uparrow +}(y_\lrcorner), \\ t^{\downarrow +} &:= \lambda x^\rho\, (t \circ x)^{\downarrow +}, \\ t^{\updownarrow -} &:= \left\langle t_\llcorner, (t_\lrcorner)^{\updownarrow -} \right\rangle . \end{aligned}$$

It is not hard to see that the two transformations are dual.

Lemma 4.16. Let A be a formula in NA^ω. Then

1. for any terms $r : \tau^+(A)$ and $s : \tau^-(A)$ we have $(r^{\uparrow +})^{\downarrow +} \overset{r}{=} r$ and $(s^{\uparrow -})^{\downarrow -} \overset{r}{=} s$,
2. for any terms $r : \sigma^*(A)$ and $s : \sigma^-(A)$ we have $(r^{\downarrow +})^{\uparrow +} \overset{r}{=} r$ and $(s^{\downarrow -})^{\uparrow -} \overset{r}{=} s$.

Proof. A syntactic exercise by induction on the definition. □

We also need to adjust the definition of the Dialectica translation accordingly.

Definition 4.17 (Quasi-linear Dialectica translation)**.** Let C be a formula in NA^ω. For $r : \sigma^*(C)$, $s : \sigma^-(C)$ we define $(\!|C|\!)^r_s$ as follows:

$$\begin{aligned} (\!|\mathrm{at}(b)|\!)^\varepsilon_\varepsilon &:= \mathrm{at}(b), \\ (\!|A \to B|\!)^r_s &:= (\!|A|\!)^{s_\llcorner}_{rs_\lrcorner} \to (\!|B|\!)^{(r \circ s_\llcorner)_\llcorner}_{s_\lrcorner}, \\ (\!|\forall x\, A|\!)^r_s &:= (\!|A\,[x := s_\llcorner]\,|\!)^{r \circ s_\llcorner}_{s_\lrcorner}. \end{aligned}$$

The relation between the original Dialectica interpretation and the quasi-linear variant is established by the following

Proposition 4.18. *Let C be a formula in* NA^ω. *Then for any terms $r : \tau^+(C)$ and $s : \tau^-(C)$, the formulas $|C|^r_s$ and $(\!|C|\!)^{r^{\uparrow +}}_{s^{\uparrow -}}$ coincide.*

Proof. Induction on the formula C.
Case $\mathrm{at}(r)$. Trivial.
Case $A \to B$. First, we note that by definition

$$\begin{aligned} s^{\uparrow -}{}_\llcorner &\overset{r}{=} (s_\llcorner)^{\uparrow +}, \\ s^{\uparrow -}{}_\lrcorner &\overset{r}{=} (s_\lrcorner)^{\uparrow -}, \\ r^{\uparrow +} s^{\uparrow -}{}_\lrcorner &\equiv (r_\lrcorner (s^{\uparrow -}{}_\llcorner)^{\downarrow +} (s^{\uparrow -}{}_\lrcorner)^{\downarrow -})^{\uparrow -} \\ &\overset{r}{=} (r_\lrcorner ((s_\llcorner)^{\uparrow +})^{\downarrow +} ((s_\lrcorner)^{\uparrow -})^{\downarrow -})^{\uparrow -} \\ \text{(by Lemma 4.16)} \qquad &\overset{r}{=} (r_\lrcorner (s_\llcorner)(s_\lrcorner))^{\uparrow -}, \\ (r^{\uparrow +} \circ s^{\uparrow -}{}_\llcorner)_\llcorner &\equiv \lambda z^{\sigma^-(B)}\, r^{\uparrow +} \left\langle (s_\llcorner)^{\uparrow +}, z \right\rangle_\llcorner \\ &\overset{r}{=} \lambda z^{\sigma^-(B)}\, (r_\llcorner ((s_\llcorner)^{\uparrow +})^{\downarrow +})^{\uparrow +} z \end{aligned}$$

$$\text{(by Lemma 4.16)} \qquad \overset{r}{=} (r_{\llcorner}(s_{\llcorner}))^{\uparrow+}.$$

Then by induction hypothesis we obtain

$$\begin{aligned}
\llparenthesis A \to B \rrparenthesis^{r^{\uparrow+}}_{s^{\uparrow-}} &= \llparenthesis A \rrparenthesis^{s^{\uparrow-}{}_{\llcorner}}_{r^{\uparrow+} s^{\uparrow-}{}_{\lrcorner}} \to \llparenthesis B \rrparenthesis^{(r^{\uparrow+} \circ s^{\uparrow-}{}_{\llcorner})_{\llcorner}}_{s^{\uparrow-}{}_{\lrcorner}} \\
&= \llparenthesis A \rrparenthesis^{(s_{\llcorner})^{\uparrow+}}_{(r_{\lrcorner}(s_{\llcorner})(s_{\lrcorner}))^{\uparrow-}} \to \llparenthesis B \rrparenthesis^{(r_{\llcorner}(s_{\llcorner}))^{\uparrow+}}_{(s_{\lrcorner})^{\uparrow-}} \\
&= |A|^{s_{\llcorner}}_{r_{\lrcorner}(s_{\llcorner})(s_{\lrcorner})} \to |B|^{r_{\llcorner}(s_{\llcorner})}_{s_{\lrcorner}} = |A \to B|^{r}_{s}\,.
\end{aligned}$$

Case $\forall x\, A$. First, we note that by definition

$$\begin{aligned}
s^{\uparrow-}{}_{\llcorner} &\overset{r}{=} s_{\llcorner}, \\
s^{\uparrow-}{}_{\lrcorner} &\overset{r}{=} (s_{\lrcorner})^{\uparrow-}, \\
r^{\uparrow+} \circ (s^{\uparrow-}{}_{\lrcorner}) &\overset{r}{=} \lambda z^{\sigma^{-}(A)}\, r^{\uparrow+} \left\langle (s_{\lrcorner})^{\uparrow-}, z \right\rangle \\
&\overset{r}{=} \lambda z^{\sigma^{-}(A)}\, (r(s_{\llcorner}))^{\uparrow+} z \overset{r}{=} (r(s_{\llcorner}))^{\uparrow+}.
\end{aligned}$$

Then by induction hypothesis we obtain

$$\begin{aligned}
\llparenthesis \forall x\, A \rrparenthesis^{r^{\uparrow+}}_{s^{\uparrow-}} &= \llparenthesis A \left[x := s^{\uparrow-}{}_{\llcorner}\right] \rrparenthesis^{r^{\uparrow+} \circ s^{\uparrow-}{}_{\lrcorner}}_{s^{\uparrow-}{}_{\lrcorner}} \\
&= \llparenthesis A \left[x := s_{\llcorner}\right] \rrparenthesis^{(r(s_{\llcorner}))^{\uparrow+}}_{(s_{\lrcorner})^{\uparrow-}} \\
&= \left|A \left[x := s_{\llcorner}\right]\right|^{r(s_{\llcorner})}_{s_{\lrcorner}} = |\forall x\, A|^{r}_{s}\,. \qquad \square
\end{aligned}$$

Corollary 4.19. *Let C be a formula in* NA^{ω}*. Then for any terms $r : \sigma^{*}(C)$, $s : \sigma^{-}(C)$, the formulas $\llparenthesis C \rrparenthesis^{r}_{s}$ and $|C|^{r^{\downarrow+}}_{s^{\downarrow-}}$ coincide.*

Proof. Follows immediately from Proposition 4.18 and Lemma 4.16. □

4.6 Soundness of the quasi-linear Dialectica interpretation

The soundness theorem for the new variant of the interpretation will follow a pattern similar to Theorem 2.21 and will be proved by induction on a proof $M : A$ with free assumption variables $u_i : C_i$. On every inductive step we will define:

1. a definition context $[\![M]\!] : \sigma^{-}(A) \Rightarrow \diamond$,
2. a context-dependent positive witnessing term $[\![M]\!]^{+} : \sigma^{+}(A)$,
3. context-dependent negative witnessing terms $[\![M]\!]^{-}_{i} : \sigma^{-}(C_i)$.

The final extracted term will be obtained by placing the context-dependent terms inside the context:

$$\{\!|M|\!\} := [\![M]\!][\langle [\![M]\!]^+, \dots, [\![M]\!]_i^-, \dots\rangle].$$

Individual components placed in the context will be referred to as follows:

$$\{\!|M|\!\}^+ := [\![M]\!][[\![M]\!]^+], \qquad \{\!|M|\!\}_i^- := [\![M]\!][[\![M]\!]_i^-].$$

Using the $\triangleright$ operation we can restore the individual parts of $\{\!|M|\!\}$:

Proposition 4.20. $\lambda y\,(\{\!|M|\!\}y \triangleright 0) \overset{r}{=} \{\!|M|\!\}^+, \lambda y\,(\{\!|M|\!\}y \triangleright i) \overset{r}{=} \{\!|M|\!\}_i^-$ *for* $y : \sigma^-(A)$.

Proof. By Proposition 4.5 we have a substitution Ξ, such that for all terms t we have $([\![M]\!]y)[t] \overset{r}{=} t\Xi$. Then we have

$$\begin{aligned} \lambda y\, \{\!|M|\!\}y \triangleright i &\equiv \lambda y\, \big(([\![M]\!]y)[\langle [\![M]\!]^+, \dots, [\![M]\!]_i^-, \dots\rangle]\big) \triangleright i \\ &\overset{r}{=} \lambda y\, \big(\langle [\![M]\!]^+, \dots, [\![M]\!]_i^-, \dots\rangle \Xi\big) \triangleright i \\ &\equiv \lambda y\, \big(\langle [\![M]\!]^+, \dots, [\![M]\!]_i^-, \dots\rangle \triangleright i\big)\Xi \\ &\overset{r}{=} \lambda y\, \big([\![M]\!]_i^- \Xi\big) \overset{r}{=} \lambda y\, \big(\{\!|M|\!\}_i^- y\big) \overset{r}{=} \{\!|M|\!\}_i^-. \end{aligned}$$ □

As with the original Dialectica interpretation, for every binary rule instance we will need to make a case distinction on the translation for every assumption variable shared between the two branches of the proof. It is easy to show that the case distinction can be implemented with a term, which has linear size on the assumption formula. We will first show that there is a term T_C, which plays the role of $((\!|C|\!)_s^r)^{\mathrm{at}}$, but its size depends linearly on the size of the formula C. Note that this linear bound does not hold directly for $((\!|C|\!)_s^r)^{\mathrm{at}}$. The reason is that even though equal subterms in extracted terms are avoided, no such claim is being made for the translation formulas. Therefore, $(\!|C|\!)_s^r$ suffers from term repetition and this will be directly transferred to its atomic translation.

Lemma 4.21. There is a constant K, such that for every formula C there is a term $T_C : \sigma^*(C) \Rightarrow \sigma^-(C) \Rightarrow \mathsf{B}$ such that:

1. $(\!|C|\!)_s^r \leftrightarrow \mathrm{at}(T_C r s)$
2. $\lceil T_C \rceil \le K \lceil C \rceil$

Proof. We define T_C by induction on the formula C.
Case $\mathrm{at}(t)$. Set $T_C := t$.
Case $A \to B$. Define

$$T_C := \lambda r\, \lambda s\, T_{\to}\big(T_A(s_{\llcorner})(r s_{\lrcorner})\big)\big(T_B((r \circ s_{\llcorner})_{\llcorner})(s_{\lrcorner})\big).$$

Since $\mathrm{at}(T_{\to}xy) \leftrightarrow (\mathrm{at}(x) \to \mathrm{at}(y))$, we can use the definition of the translation to show the correctness of T_C. Moreover, we have $\lceil T_{\to} \rceil = 8$ and thus $\lceil T_C \rceil \leq \lceil T_A \rceil + \lceil T_B \rceil + 32$.

Case $\forall x\, A$. Define

$$T_C := \lambda r\, \lambda s\, \mathbf{let}\ x := s_{\llcorner}\ \mathbf{in}\ T_A(r(s_{\llcorner}))(s_{\lrcorner}), \text{ and we have}$$
$$\lceil T_C \rceil \leq \lceil T_A \rceil + 14.$$

It is straightforward to check the correctness of T_C. □

A drawback of the original Dialectica interpretation is that if the assumption variable has n occurrences, a case distinction for the same formula can be repeated $n-1$ times in the extracted term. Thus it is more efficient that for every proof $\mathcal{P}$ we put the extracted terms in a definition context $D : \diamond$ containing the definitions of $T_{\to}$ and all T_C, such that $u : C$ is an assumption variable of M. It is clear that $\lceil D \rceil$ is bounded by the sum of the size of all assumption formulas, which by Definition 4.9 is definitely not greater than $\lceil M \rceil$. In order to keep the presentation simpler, in the following we will not be explicit about the context D; we will just assume that it is the outermost context of the final extracted term and that we have access to variables d_C instantiated with T_C in D.

Lemma 4.22 (Linear Dialectica case distinction)**.** Let C be a formula, let $x : \sigma^*(C)$ be a variable and let $t_1, t_2 : \sigma^-(C)$ be terms. Let D be a definition context associating a variable d_C with the term T_C from Lemma 4.21. Then there is a term t such that

1. $\vdash (\!|C|\!)^x_{D[t]} \to (\!|C|\!)^x_{t_i}$ for $i = 1, 2$,
2. $\lceil t \rceil \leq \lceil t_1 \rceil + \lceil t_2 \rceil + K$, where K is a constant independent of the formula C.

Proof. Similarly to Lemma 2.19 we define

$$t_1 \overset{C,x}{\bowtie} t_2 := \begin{cases} t_1, & \text{if } t_1 \equiv t_2, \\ \mathbf{let}\ y := t_1\ \mathbf{in}\ \mathsf{Cases}\,(d_C x y) t_2 y, & \text{otherwise.} \end{cases}$$

Assuming that by Lemma 4.21 we have proof terms $K : \mathrm{at}(T_C x t_1) \to (\!|C|\!)^x_{t_1}$ and $L : (\!|C|\!)^x_{t_1} \to \mathrm{at}(T_C x t_1)$, we can define the proof terms $\mathcal{Q}_i : (\!|C|\!)^x_{D[t_1 \overset{C,x}{\bowtie} t_2]} \to (\!|C|\!)^x_{t_i}$ exactly as in Lemma 2.19. Moreover, $\lceil t_1 \overset{C,x}{\bowtie} t_2 \rceil \leq \lceil t_1 \rceil + \lceil t_2 \rceil + 12$. □

Theorem 4.23 (Soundness of quasi-linear Dialectica interpretation)**.** *Let $A \in \mathrm{NA}^{\omega}$ be a formula and let $\mathcal{P}^A$ be a proof term with assumptions among $\{u_i : C_i\}_{i \geq 1}$. Let us have fresh witnessing variables $X = \{x_i : \sigma^*(C_i)\}$, each one associated uniquely with an assumption variable u_i and let $y_A : \sigma^-(A)$ be a fresh challenging variable associated* *uniquely with the formula A. Then there is a term $\{\!|\mathcal{P}|\!\}$ and a proof $\overline{\mathcal{P}} : (\!|A|\!)^{\{\!|\mathcal{P}|\!\}^+}_{y_A}$, such that*

1. $\mathsf{FA}(\overline{\mathcal{P}}) \subseteq \left\{ v_i : (\!|C_i|\!)^{x_i}_{\{\!|\mathcal{P}|\!\}^-_i\, y_A} \right\}$*, where each v_i is associated with the corresponding u_i,*
2. $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) \subseteq \mathsf{FV}(\mathcal{P}) \cup X$,
3. $\lceil \{\!|\mathcal{P}|\!\} \rceil \leq K(\lceil \mathcal{P} \rceil \llbracket \mathcal{P} \rrbracket^2)$ *for a fixed constant K, not depending on $\mathcal{P}$.*

Proof. We prove the theorem by induction on $\mathcal{P}$.

Case u_1^A. Set $\llbracket \mathcal{P} \rrbracket := \lambda y_A\, []$, $\llbracket \mathcal{P} \rrbracket^+ := x_1 y_A$, $\llbracket \mathcal{P} \rrbracket^-_1 := y_A$. Then $\{\!|\mathcal{P}|\!\}^+ \equiv \lambda y_A\, x_1 y_A \overset{r}{=} x_1$ and $\{\!|\mathcal{P}|\!\}^-_i\, y_A \overset{r}{=} y_A$, thus we can set $\overline{\mathcal{P}} := v_1$. The variable conditions are obviously satisfied and $\lceil \{\!|\mathcal{P}|\!\} \rceil \leq 6$.

Case $\lambda u_0^B\, M^C$. By induction hypothesis we have a proof term $\overline{M} : (\!|C|\!)^{\{\!|M|\!\}^+}_{y_C}$ with assumptions $w_0 : (\!|B|\!)^{x_0}_{\{\!|M|\!\}^-_0\, y_C}$ and $w_i : (\!|C_i|\!)^{x_i}_{\{\!|M|\!\}^-_i\, y_C}$ for $i \geq 1$. Define

$$\begin{aligned}
\llbracket \mathcal{P} \rrbracket\ \ &:= \lambda^\circ x_0\, \llbracket M \rrbracket, \\
\llbracket \mathcal{P} \rrbracket^+ &:= \left\langle \llbracket M \rrbracket^+, \llbracket M \rrbracket^-_0 \right\rangle, \\
\llbracket \mathcal{P} \rrbracket^-_i &:= \llbracket M \rrbracket^-_i .
\end{aligned}$$

The variable conditions is satisfied, since $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) = \mathsf{FV}(\{\!|M|\!\}) \setminus \{x_0\}$. We will use the substitution $\xi := [x_0 := y_A\llcorner]\, [y_C := y_A\lrcorner]$. Since

$$\begin{aligned}
(\!|B \to C|\!)^{\{\!|\mathcal{P}|\!\}^+}_{\langle x_0, y_C \rangle} &= (\!|B|\!)^{x_0}_{\{\!|\mathcal{P}|\!\}^+ \langle x_0, y_C \rangle \lrcorner} \to (\!|C|\!)^{\lambda y_C\, \{\!|\mathcal{P}|\!\}^+ \langle x_0, y_C \rangle \llcorner}_{y_C} \\
&= (\!|B|\!)^{x_0}_{\{\!|M|\!\}^-_0\, y_C} \to (\!|C|\!)^{\lambda y_C\, \{\!|M|\!\}^+ y_C}_{y_C}, \text{ and} \\
(\!|C_i|\!)^{x_i}_{\{\!|\mathcal{P}|\!\}^-_i \langle x_0, y_C \rangle} &= (\!|C_i|\!)^{x_i}_{\{\!|M|\!\}^-_i\, y_C},
\end{aligned}$$

we can set $\overline{\mathcal{P}} := (\lambda w_0\, \overline{M})\xi$, with $v_i := w_i \xi$. We can also see that $\lceil \{\!|\mathcal{P}|\!\} \rceil \leq \lceil \{\!|M|\!\} \rceil + 9$.

Case $M_1^{C \to A} M_2^C$. Let us denote $B := C \to A$. By induction hypothesis we have proofs

$$\begin{aligned}
&\overline{M}_1 : (\!|B|\!)^{\{\!|M_1|\!\}^+}_{y_B} \text{ with assumptions } w'_i : (\!|C_i|\!)^{x_i}_{\{\!|M_1|\!\}^-_i\, y_B} \text{ and} \\
&\overline{M}_2 : (\!|C|\!)^{\{\!|M_2|\!\}^+}_{y_C} \text{ with assumptions } w''_i : (\!|C_i|\!)^{x_i}_{\{\!|M_2|\!\}^-_i\, y_C}.
\end{aligned}$$

We will use the substitutions

$$\begin{aligned}
\xi_1 &:= \left[y_B := \left\langle \{\!|M_2|\!\}^+, y_A \right\rangle \right] && \text{for } M_1, \\
\xi_2 &:= \left[y_C := \{\!|M_1|\!\}^+ \left\langle \{\!|M_2|\!\}^+, y_A \right\rangle \lrcorner \right] && \text{for } M_2.
\end{aligned}$$

By unfolding the definition of the translation we obtain:

$$\overline{M_1}\xi_1 : \llparenthesis C \rrparenthesis^{\{\!|M_2|\!\}^+}_{\{\!|M_1|\!\}^+\left\langle\{\!|M_2|\!\}^+, y_A\right\rangle\lrcorner} \to \llparenthesis A \rrparenthesis^{(\{\!|M_1|\!\}^+\circ\{\!|M_2|\!\}^+)\llcorner}_{y_A} \text{ with } w_i'\xi_1 : \llparenthesis C_i \rrparenthesis^{x_i}_{\{\!|M_1|\!\}^-_i\left\langle\{\!|M_2|\!\}^+, y_A\right\rangle},$$
$$\overline{M_2}\xi_2 : \llparenthesis C \rrparenthesis^{\{\!|M_2|\!\}^+}_{\{\!|M_1|\!\}^+\left\langle\{\!|M_2|\!\}^+, y_A\right\rangle\lrcorner} \text{ with } w_i''\xi_2 : \llparenthesis C_i \rrparenthesis^{x_i}_{\{\!|M_2|\!\}^-_i(\{\!|M_1|\!\}^+\left\langle\{\!|M_2|\!\}^+, y_A\right\rangle\lrcorner)}.$$

For f and z fresh variables we define

$$\begin{aligned}
\llbracket\mathcal{P}\rrbracket &:= \textbf{let } f := \{\!|M_2|\!\} \textbf{ in } \llbracket M_1\rrbracket \circ (f\llcorner)[\textbf{let } z := \llbracket M_1\rrbracket^+ \textbf{ in } []],\\
\llbracket\mathcal{P}\rrbracket^+ &:= z\llcorner,\\
\llbracket\mathcal{P}\rrbracket^-_i &:= \llbracket M_1\rrbracket^-_i \overset{C_i}{\bowtie} f(z\lrcorner) \triangleright i
\end{aligned}$$

Using Proposition 4.20 we obtain

$$\begin{aligned}
\{\!|\mathcal{P}|\!\}^+ \quad &\overset{r}{=} \llbracket M_1\rrbracket \circ \{\!|M_2|\!\}^+[\llbracket M_1\rrbracket^+\llcorner] \overset{r}{=} (\{\!|M_1|\!\}^+ \circ \{\!|M_2|\!\}^+)\llcorner,\\
\{\!|\mathcal{P}|\!\}^-_i y_A &\overset{r}{=} \llbracket M_1\rrbracket \left\langle\{\!|M_2|\!\}^+, y_A\right\rangle [\llbracket M_1\rrbracket^-_i \overset{C_i}{\bowtie} \{\!|M_2|\!\}^-_i(\llbracket M_1\rrbracket^+\lrcorner)].
\end{aligned}$$

Let us define $t_1 := \llbracket M_1\rrbracket^-_i$, $t_2 := \{\!|M_2|\!\}^-_i(\llbracket M_1\rrbracket^+\lrcorner)$ and $E := \llbracket M_1\rrbracket \left\langle\{\!|M_2|\!\}^+, y_A\right\rangle$ so that the term $\{\!|\mathcal{P}|\!\}^-_i y_A$ can be written as $E[t_1 \overset{C_i}{\bowtie} t_2]$. As before, we assume that $\llbracket M_j\rrbracket^-_i := \llbracket M_{3-j}\rrbracket^-_i$ whenever $u_i \in \mathsf{FA}(M_{3-j}) \setminus \mathsf{FA}(M_j)$. By Lemma 4.22 we have proof terms $\mathcal{Q}^{(j)}_i : \llparenthesis C_i \rrparenthesis^{x_i}_{t_1 \overset{i}{\bowtie} t_2} \to \llparenthesis C_i \rrparenthesis^{x_i}_{t_j}$ for $j = 1, 2$. By Proposition 4.5 we have a substitution Ξ, which is such that $E[t] \overset{r}{=} t\Xi$. Thus we obtain $\mathcal{Q}^{(j)}_i\Xi : \llparenthesis C_i \rrparenthesis^{x_i}_{E[t_1 \overset{i}{\bowtie} t_2]} \to \llparenthesis C_i \rrparenthesis^{x_i}_{E[t_j]}$. On the other hand

$$\begin{aligned}
E[t_1] \equiv \llbracket M_1\rrbracket \left\langle\{\!|M_2|\!\}^+, y_A\right\rangle [\llbracket M_1\rrbracket^-_i] \qquad\qquad\qquad &\overset{r}{=} \{\!|M_1|\!\}^-_i \left\langle\{\!|M_2|\!\}^+, y_A\right\rangle,\\
E[t_2] \equiv \llbracket M_1\rrbracket \left\langle\{\!|M_2|\!\}^+, y_A\right\rangle [\{\!|M_2|\!\}^-_i(\llbracket M_1\rrbracket^+\lrcorner)] &\overset{r}{=} \{\!|M_2|\!\}^-_i \llbracket M_1\rrbracket \left\langle\{\!|M_2|\!\}^+, y_A\right\rangle [\llbracket M_1\rrbracket^+\lrcorner]\\
&\overset{r}{=} \{\!|M_2|\!\}^-_i(\{\!|M_1|\!\}^+ \left\langle\{\!|M_2|\!\}^+, y_A\right\rangle\lrcorner),
\end{aligned}$$

where for the last equality we used Proposition 4.20 and Corollary 4.6, assuming that the set $\mathsf{BV}(\llbracket M_1\rrbracket)$ consists of fresh variables and thus has an empty intersection with $\mathsf{FV}(\{\!|M_2|\!\}^-_i)$. Then we can define as in Theorem 2.21

$$\overline{\mathcal{P}} := \overline{\mathcal{P}}_1\overline{\mathcal{P}}_2, \qquad \text{where } \overline{\mathcal{P}}_j := \overline{M}_j\xi_j\vec{\eta}_j \text{ with } \eta_{j,i} = \left[w^{(j)}_i := \mathcal{Q}^{(j)}_i\Xi v_i\right].$$

The variable condition is obviously satisfied as $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) \subseteq \mathsf{FV}(\{\!|M_1|\!\}) \cup \mathsf{FV}(\{\!|M_2|\!\})$. The size of the extracted terms is calculated as follows:

$$\begin{aligned}
\lceil\llbracket\mathcal{P}\rrbracket\rceil \quad &\leq \lceil\{\!|M_2|\!\}\rceil + \lceil\llbracket M_1\rrbracket\rceil + \lceil\llbracket M_1\rrbracket^+\rceil + 9,\\
\lceil\llbracket\mathcal{P}\rrbracket^+\rceil &\leq 2,\\
\lceil t_1 \overset{C_i}{\bowtie} t_2\rceil &\leq \lceil t_1\rceil + \lceil t_2\rceil + 12,
\end{aligned}$$

$$\lceil \llbracket \mathcal{P} \rrbracket_i^- \rceil \quad \leq \lceil \llbracket M_1 \rrbracket_i^- \rceil + i + 17 \leq \lceil \llbracket M_1 \rrbracket_i^- \rceil + \llbracket \mathcal{P} \rrbracket + 17,$$

hence

$$\lceil \{\!\!| \mathcal{P} |\!\!\} \rceil \quad \leq \lceil \{\!\!| M_1 |\!\!\} \rceil + \lceil \{\!\!| M_2 |\!\!\} \rceil + \llbracket \mathcal{P} \rrbracket^2 + 17 \llbracket \mathcal{P} \rrbracket + 23,$$

because the number of free assumptions C_i in the proof $\mathcal{P}$ is bounded by $\llbracket \mathcal{P} \rrbracket$.

Case $\lambda x^\rho M^B$. By induction hypothesis we have a proof of $\overline{M} : (\!| B |\!)_{y_B}^{\{\!|M|\!\}^+}$ with assumptions $w_i : (\!| C_i |\!)_{\{\!|M|\!\}_i^- y_B}^{x_i}$. Define

$$\llbracket \mathcal{P} \rrbracket := \lambda^\circ x \, \llbracket M \rrbracket, \qquad \llbracket \mathcal{P} \rrbracket^+ := \llbracket M \rrbracket^+, \qquad \llbracket \mathcal{P} \rrbracket_i^- := \llbracket M \rrbracket_i^-.$$

Since $A = \forall x \, B$, we can substitute $\xi := [x := y_A \llcorner] \, [y_B := y_A \lrcorner]$. Since by definition

$$\begin{aligned} (\!| \forall x \, B |\!)_{\langle x, y_B \rangle}^{\{\!|\mathcal{P}|\!\}^+} &= (\!| B |\!)_{y_B}^{\{\!|\mathcal{P}|\!\}^+ \circ x} = (\!| B |\!)_{y_B}^{\lambda y_B \, \{\!|M|\!\}^+ y_B}, \text{and} \\ (\!| C_i |\!)_{\{\!|\mathcal{P}|\!\}_i^- \langle x, y_B \rangle}^{x_i} &= (\!| C_i |\!)_{\{\!|M|\!\}_i^- y_B}^{x_i}, \end{aligned}$$

we can define $\overline{\mathcal{P}} := \overline{M}\xi$ with $v_i := w_i \xi$. The variable condition still holds since $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) := \mathsf{FV}(\{\!|M|\!\}) \setminus \{x\}$. Moreover, $\{\!|\mathcal{P}|\!\} = \{\!|M|\!\} + 8$.

Case $M^{\forall x^\rho B} t^\rho$. Let $C := \forall x \, B$. By induction hypothesis we have a proof

$$\overline{M} : (\!| \forall x \, B |\!)_{y_C}^{\{\!|M|\!\}^+} \qquad \text{with } w_i : (\!| C_i |\!)_{\{\!|M|\!\}_i^- y_C}^{x_i},$$

which, after applying the substitution $\xi := [y_C := \langle t, y_A \rangle]$ and unfolding the definition becomes

$$\overline{M}\xi : (\!| A |\!)_{y_A}^{\{\!|M|\!\}^+ \circ t} \qquad \text{with } w_i \xi : (\!| C_i |\!)_{\{\!|M|\!\}_i^- \langle t, y_A \rangle}^{x_i}.$$

Then it becomes clear that we can define

$$\begin{aligned} \llbracket \mathcal{P} \rrbracket &:= \llbracket M \rrbracket \circ t, & \llbracket \mathcal{P} \rrbracket^+ &:= \llbracket M \rrbracket^+, \\ \overline{\mathcal{P}} &:= \overline{M}\xi \text{ with } v_i := w_i \xi, & \llbracket \mathcal{P} \rrbracket_i^- &:= \llbracket M \rrbracket_i^-. \end{aligned}$$

The variable condition holds since $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) = \mathsf{FV}(\{\!|M|\!\}) \cup \mathsf{FV}(t)$. Finally, $\lceil \{\!|\mathcal{P}|\!\} \rceil \leq \lceil \{\!|M|\!\} \rceil + \lceil t \rceil + 4$.

Case AxT. Trivial as in Theorem 2.21.

Case $\mathcal{C}^{b,A} \, b M_{\mathsf{tt}}^{A[b:=\mathsf{tt}]} M_{\mathsf{ff}}^{A[b:=\mathsf{ff}]}$. By induction hypothesis we have proofs

$$\overline{M}_j : (\!| A \, [b := j] \, |\!)_{y_A}^{\{\!|M_j|\!\}^+} \text{ with assumptions } w_i^{(j)} : (\!| C_i |\!)_{\{\!|M_j|\!\}_i^- y_A}^{x_i} \text{ for } j = \mathsf{ff}, \mathsf{tt}.$$

Let us define

$$\begin{aligned} \llbracket\mathcal{P}\rrbracket &:= \lambda y_A \,\llbracket M_{\mathsf{tt}}\rrbracket y_A[\llbracket M_{\mathsf{ff}}\rrbracket y_A], \\ \llbracket\mathcal{P}\rrbracket^+ &:= \mathsf{Cases}\, b\, \llbracket M_{\mathsf{tt}}\rrbracket^+ \,\llbracket M_{\mathsf{ff}}\rrbracket^+, \\ \llbracket\mathcal{P}\rrbracket^- &:= \mathsf{Cases}\, b\, \llbracket M_{\mathsf{tt}}\rrbracket^-_i \,\llbracket M_{\mathsf{ff}}\rrbracket^-_i . \end{aligned}$$

Assuming that the bound variables of $\llbracket M_{\mathsf{tt}}\rrbracket$ and $\llbracket M_{\mathsf{ff}}\rrbracket$ are unique, using Proposition 4.5 we can show that $\{\!|\mathcal{P}|\!\}\,[b := j] \overset{r}{=} \{\!|M_j|\!\}$ for $j = \mathsf{tt}, \mathsf{ff}$. Then, as in Theorem 2.21 we can define $\overline{\mathcal{P}} := \mathcal{C}\, b\, (\lambda\vec{w}'\, \overline{M}_{\mathsf{tt}})\, (\lambda\vec{w}''\, \overline{M}_{\mathsf{ff}})\vec{v}$. The variable conditions are satisfied, because $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) = \mathsf{FV}(\{\!|M|\!\}) \cup \mathsf{FV}(\{\!|N|\!\}) \cup \{b\}$. We also have $\lceil\{\!|\mathcal{P}|\!\}\rceil \le \lceil\{\!|M_{\mathsf{tt}}|\!\}\rceil + \lceil\{\!|M_{\mathsf{ff}}|\!\}\rceil + 5\llbracket\mathcal{P}\rrbracket + 9$.

Case $\mathsf{Ind}_{\mathsf{N}}^{n,A}\, n\, M_1^{A[n:=0]}\, (\lambda n\, \lambda u_0^A\, M_2^{A[n:=\mathsf{S}n]})$. By induction hypothesis we have proofs

$$\begin{aligned} &\overline{M}_1 : (\!| A\,[n := 0]\, |\!)_{y_A}^{\{\!|M_1|\!\}^+} \text{ with assumptions } && w_i' : (\!|C_i|\!)^{x_i}_{\{\!|M_1|\!\}_i^- y_A} \text{ for } i \ge 1 \text{ and} \\ &\overline{M}_2 : (\!| A\,[n := \mathsf{S}n]\, |\!)_{y_A}^{\{\!|M_2|\!\}^+} \text{ with assumptions } && w_0'' : (\!|A|\!)^{x_0}_{\{\!|M_2|\!\}_0^- y_A} \text{ and} \\ & && w_i'' : (\!|C_i|\!)^{x_i}_{\{\!|M_2|\!\}_i^- y_A} \text{ for } i \ge 1. \end{aligned}$$

As before, for the sake of unified treatment let us define $\llbracket M_j\rrbracket^-_i := \llbracket M_{3-j}\rrbracket^-_i$ if $u_i \in \mathsf{FV}(M_{3-j}) \setminus \mathsf{FV}(M_j)$ for $i \ge 1$. Take a fresh variable z and consider the terms

$$\begin{aligned} \llbracket L\rrbracket &:= \mathcal{R}_{\mathsf{N}}\, n\, \{\!|M_1|\!\}\big(\lambda n\, \lambda p\, \mathbf{let}\ x_0 := p_{\llcorner}\ \mathbf{in}\ \llbracket M_2\rrbracket[\mathbf{let}\ z := p\llbracket M_2\rrbracket_0^-\ \mathbf{in}\ []]\big), \\ \llbracket L\rrbracket^+ &:= \llbracket M_2\rrbracket^+, \\ \llbracket L\rrbracket^-_i &:= \llbracket M_2\rrbracket^-_i \overset{C_i}{\bowtie} (z \triangleright i). \end{aligned}$$

It seems like $\{\!|L|\!\}$ is the needed extracted term, but there is a subtle problem: $\llbracket L\rrbracket$ is a term context, but is not a definition context! Fortunately, this can be repaired by taking a fresh variable a and defining:

$$\begin{aligned} \llbracket\mathcal{P}\rrbracket &:= \lambda y_A\, \mathbf{let}\ a := \{\!|L|\!\} y_A\ \mathbf{in}\ [], \\ \llbracket\mathcal{P}\rrbracket^+ &:= a_{\llcorner}, \\ \llbracket\mathcal{P}\rrbracket^-_i &:= a \triangleright i. \end{aligned}$$

Thus $\{\!|\mathcal{P}|\!\}^+ \overset{r}{=} \{\!|L|\!\}_{\llcorner}$ and $\{\!|\mathcal{P}|\!\}^-_i y_A \overset{r}{=} \{\!|L|\!\} y_A \triangleright i$. By unfolding the definition and applying Corollary 4.6 and Proposition 4.20 to $\llbracket M_2\rrbracket$ we obtain

$$\begin{aligned} &\{\!|L|\!\}\,[n := 0] && \overset{r}{=} \{\!|M_1|\!\} && (*) \\ &\{\!|L|\!\}\,[n := \mathsf{S}n]_{\llcorner} && \overset{r}{=} \mathbf{let}\ x_0 := \{\!|L|\!\}_{\llcorner}\ \mathbf{in}\ \{\!|M_2|\!\}^+, \\ &\{\!|L|\!\}\,[n := \mathsf{S}n]\, y_A \triangleright i && \overset{r}{=} \mathbf{let}\ x_0 := \{\!|L|\!\}_{\llcorner}\ \mathbf{in}\ \{\!|M_2|\!\}^-_i y_A \overset{C_i}{\bowtie} \{\!|L|\!\}(\{\!|M_2|\!\}^-_0 y_A) \triangleright i. \end{aligned}$$

Following the argument in Theorem 2.21, we will define a proof $\mathcal{Q}$ of the formula $B := \forall y_A\, (\vec{D} \to (\!|A|\!)_{y_A}^{\{\!|\mathcal{P}|\!\}^+})$, where $D_i := (\!|C_i|\!)^{x_i}_{\{\!|\mathcal{P}|\!\}_i^- y_A}$. Then we will be able to set $\overline{\mathcal{P}} := \mathcal{Q} y_A \vec{v}$. By definition

$$B\,[n := 0] = \forall y_A \left(\overrightarrow{(\!|C_i|\!)^{x_i}_{\{\!|M_1|\!\}_i^- y_A}} \to (\!|A\,[n := 0]\,|\!)_{y_A}^{\{\!|M_1|\!\}^+}\right), \text{ which is proved by } \overline{M}_1,$$
$$B\,[n := \mathsf{S}n] = \forall y_A \left(\overrightarrow{(\!|C_i|\!)^{x_i}_{t_i \xi}} \to (\!|A\,[n := \mathsf{S}n]\,|\!)_{y_A}^{\{\!|M_2|\!\}^+ \xi}\right), \text{ where}$$

$\xi := \left[x_0 := \{\!|\mathcal{P}|\!\}^+\right]$, $t_i := t_i' \overset{C_i}{\bowtie} t_i''$ with $t_i' := \{\!|M_2|\!\}_i^- y_A$, $t_i'' := \{\!|L|\!\}(\{\!|M_2|\!\}_0^- y_A) \triangleright i$. By Lemma 2.19 we have proofs $\mathcal{Q}_i^{(j)} : (\!|C_i|\!)^{x_i}_{t_i \xi} \to (\!|C_i|\!)^{x_i}_{t_i^{(j)} \xi}$ and $\mathcal{Q}$ is defined by induction in a similar fashion as in Theorem 2.21:

$$\mathcal{Q} := \mathsf{Ind}_{\mathsf{N}}^{n,B}\, n\, (\lambda y_A\, \lambda \vec{w}'\, \overline{M}_1)(\lambda n\, \lambda p^B\, \lambda y_A\, \lambda \vec{v}\, (\lambda \vec{w}''\, \overline{M}_2) \xi (p \{\!|M_2|\!\}_0^- \overrightarrow{\mathcal{Q}_i' v_i}) \overrightarrow{\mathcal{Q}_i'' v_i}).$$

The variable conditions hold because $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) \subseteq \mathsf{FV}(\{\!|M_1|\!\}) \cup \mathsf{FV}(\{\!|M_2|\!\}) \setminus \{x_0\} \cup \{n\}$ For the size of the extracted terms we obtain:

$$\begin{array}{rl}
& \lceil \llbracket L \rrbracket^+ \rceil = \lceil \llbracket M_2 \rrbracket^+ \rceil \\
& \lceil \llbracket L \rrbracket^- \rceil \leq \lceil \llbracket M_2 \rrbracket_i^- \rceil + i + 14 \leq \lceil \llbracket M_2 \rrbracket_i^- \rceil + \llbracket \mathcal{P} \rrbracket + 14 \\
& \lceil \llbracket L \rrbracket \rceil \leq \lceil \{\!|M_1|\!\} \rceil + \lceil \llbracket M_2 \rrbracket_0^- \rceil + \lceil \llbracket M_2 \rrbracket \rceil + 15, \\
\text{hence} & \lceil \{\!|L|\!\} \rceil \leq \lceil \{\!|M_1|\!\} \rceil + \lceil \{\!|M_2|\!\} \rceil + \llbracket \mathcal{P} \rrbracket^2 + 15 \llbracket \mathcal{P} \rrbracket + 14, \\
& \lceil \llbracket \mathcal{P} \rrbracket \rceil \leq \lceil \{\!|L|\!\} \rceil + 6, \\
& \lceil \llbracket \mathcal{P} \rrbracket^+ \rceil \leq 2, \\
& \lceil \llbracket \mathcal{P} \rrbracket_i^- \rceil \leq i + 2 \leq \llbracket \mathcal{P} \rrbracket + 2 \\
\text{hence} & \lceil \{\!|\mathcal{P}|\!\} \rceil \leq \lceil \{\!|M_1|\!\} \rceil + \lceil \{\!|M_2|\!\} \rceil + 2 \llbracket \mathcal{P} \rrbracket^2 + 18 \llbracket \mathcal{P} \rrbracket + 22.
\end{array}$$

Case $\mathsf{Ind}_{\mathsf{L}(\rho)}^{l,A}\, l\, M_1^{A[l:=\mathsf{nil}]}\, (\lambda x\, \lambda l\, \lambda u_0^A\, M_2^{A[l:=x::l]})$. This case is very similar to the previous one. By induction hypothesis we have proofs

$$\begin{array}{ll}
\overline{M}_1 : (\!|A\,[l := \mathsf{nil}]\,|\!)_{y_A}^{\{\!|M_1|\!\}^+} \text{ with assumptions} & w_i' : (\!|C_i|\!)^{x_i}_{\{\!|M_1|\!\}_i^- y_A} \text{ for } i \geq 1 \text{ and} \\
\overline{M}_2 : (\!|A\,[l := x :: l]\,|\!)_{y_A}^{\{\!|M_2|\!\}^+} \text{ with assumptions} & w_0'' : (\!|A|\!)^{x_0}_{\{\!|M_2|\!\}_0^- y_A} \text{ and} \\
& w_i'' : (\!|C_i|\!)^{x_i}_{\{\!|M_2|\!\}_i^- y_A} \text{ for } i \geq 1.
\end{array}$$

Let us define

$$\llbracket L \rrbracket \ := \mathcal{R}_{\mathsf{L}(\rho)}\, l\, \{\!|M_1|\!\} \big(\lambda x\, \lambda l\, \lambda p\, \textbf{let } x_0 := p_{\llcorner} \textbf{ in } \llbracket M_2 \rrbracket [\textbf{let } z := p \llbracket M_2 \rrbracket_0^- \textbf{ in } [\,]]\big),$$
$$\llbracket L \rrbracket^+ := \llbracket M_2 \rrbracket^+, \qquad \llbracket L \rrbracket_i^- := \llbracket M_2 \rrbracket_i^- \overset{C_i}{\bowtie} (z \triangleright i),$$

$$\begin{aligned} \llbracket \mathcal{P} \rrbracket &:= \lambda y_A \, \mathbf{let} \ a := \{\!| L |\!\} y_A \ \mathbf{in} \ [], \\ \llbracket \mathcal{P} \rrbracket^+ &:= a_{\mathsf{L}}, \qquad\qquad \llbracket \mathcal{P} \rrbracket^-_i := a \triangleright i. \end{aligned}$$

We adopt the definitions of D_i, t_i, $t_i^{(j)}$ and $\mathcal{Q}_i^{(j)}$ from the previous case and, as before, set $\overline{\mathcal{P}} := \mathcal{Q} y_A \vec{v}$, where the proof $\mathcal{Q}$ of the formula $B := \forall y_A \, (\vec{D} \to (\!| A |\!)_{y_A}^{\{\!|\mathcal{P}|\!\}^+})$ is defined as

$$\mathcal{Q} := \mathsf{Ind}^{l,B}_{\mathsf{L}(\rho)} \, l \, (\lambda y_A \, \lambda \vec{w}' \, \overline{M}_1)(\lambda x \, \lambda l \, \lambda p^B \, \lambda y_A \, \lambda \vec{v} \, (\lambda \vec{w}'' \, \overline{M}_2) \xi (p \{\!| M_2 |\!\}^-_0 \overrightarrow{\mathcal{Q}'_i v_i}) \overrightarrow{\mathcal{Q}''_i v_i}).$$

The variable conditions and the size bounds hold as in the previous case. □

Remark 4.24. Formally, the bound we have obtained in Theorem 4.23 is not linear, as it depends quadratically on the measure $\llbracket \mathcal{P} \rrbracket$, which in the worst case could be equal to $\lceil \mathcal{P} \rceil$. However, as $\lceil \mathcal{P} \rceil$ increases, $\llbracket \mathcal{P} \rrbracket$ grows much slower, and hence for practical cases, the size of extracted terms can be considered as "almost" linear in the size of the proof.

The quadratic dependency on the parameter $\llbracket \mathcal{P} \rrbracket$ is caused by the technical subtleties related to "unpacking" the terms $\{\!| \mathcal{P} |\!\}$ using the selectors $\{\!| \mathcal{P} |\!\} \triangleright i$. In case we work in a term language equipped with means for constructing polymorphic arrays of the type of $\{\!| \mathcal{P} |\!\}$, for which the components can be randomly accessed in constant time, we can improve the size bound to $O(\lceil \mathcal{P} \rceil \llbracket \mathcal{P} \rrbracket)$. The overhead $\llbracket \mathcal{P} \rrbracket$ cannot be completely avoided if we insist on working in a convenient natural deduction settings, where the assumptions appear as separate entities. In any case, the construction of such a language goes beyond the scope of the current work, which aims to stay theoretically as close as possible to a purely functional language in the spirit of Gödel's system T.

Remark 4.25. The size bound $O(\lceil \mathcal{P} \rceil + \llbracket \mathcal{P} \rrbracket^2)$, which was claimed in [Tri10b], was incorrectly calculated. In the worst case the term size increases by $\llbracket \mathcal{P} \rrbracket^2$ on *every inductive step*, and since the induction steps are linear in the size of the proof, the correct estimation is $O(\lceil \mathcal{P} \rceil \llbracket \mathcal{P} \rrbracket^2)$.

Remark 4.26. A worst case scenario for Theorem 4.23 can be constructed as follows. Let A and B be arbitrary atomic formulas. Let $u_0 : \forall x \, A$, $u_{2n+1} : \forall x \, A \to \forall y \, B$ and $u_{2n+2} : \forall y \, B \to \forall x \, A$ be assumption variables. We define the sequence of proofs $\mathcal{P}_n$ such that $\mathcal{P}_{2n} : \forall x \, A$ and $\mathcal{P}_{2n+1} : \forall y \, B$ as follows:

$$\mathcal{P}_0 := u_0, \qquad \mathcal{P}_{n+1} := u_{n+1} \mathcal{P}_n.$$

Let $\lceil u_{n+1} \rceil = K$ and $\lceil u_0 \rceil = L$ for fixed constants K and L, depending on the sizes of the formulas A and B. Then $\lceil \mathcal{P}_n \rceil := n(K+1) + L$ and hence is $O(n)$. The extracted terms from the proofs $\mathcal{P}_n$ are defined as in Theorem 4.23:

$$\begin{aligned}
[\![\mathcal{P}_0]\!] &\equiv \lambda y_0\, [],\\
[\![\mathcal{P}_0]\!]^+ &\equiv x_0 y_0,\\
[\![\mathcal{P}_0]\!]^-_{u_0} &\equiv y_0,\\
[\![\mathcal{P}_{n+1}]\!] &\equiv \textbf{let } f_n := \{\!|\mathcal{P}_n|\!\} \textbf{ in let } y_{n+1} := f_{n\llcorner} \textbf{ in let } z_n := x_{n+1}y_{n+1} \textbf{ in } [],\\
[\![\mathcal{P}_{n+1}]\!]^+ &\equiv z_{n\llcorner},\\
[\![\mathcal{P}_{n+1}]\!]^-_a &\equiv \begin{cases} y_{n+1}, & \text{if } a = n+1,\\ f_n(z_{n\lrcorner}) \rhd a, & \text{otherwise.} \end{cases}
\end{aligned}$$

Their size is calculated as follows:

$$\begin{aligned}
\lceil \{\!|\mathcal{P}_0|\!\} \rceil &= 6,\\
\lceil [\![\mathcal{P}_{n+1}]\!] \rceil &= \lceil \{\!|\mathcal{P}_n|\!\} \rceil + 12,\\
\lceil [\![\mathcal{P}_{n+1}]\!]^+ \rceil &= 2,\\
\lceil [\![\mathcal{P}_{n+1}]\!]^-_a \rceil &= \begin{cases} 1, & \text{if } a = n+1,\\ a+6, & \text{otherwise,} \end{cases}\\
\lceil \{\!|\mathcal{P}_{n+1}|\!\} \rceil &= \lceil \{\!|\mathcal{P}_n|\!\} \rceil + \frac{n(n+1)}{2} + 6n + 22, \text{ hence}\\
\lceil \{\!|\mathcal{P}_n|\!\} \rceil &= \frac{(n-1)n(2n-1)}{12} + 13\frac{n(n-1)}{24} + 22n + 6, \text{ which is } O(n^3).
\end{aligned}$$

Corollary 4.27 (Quasi-linear Dialectica extraction)**.** *Let $\mathcal{P} : C$ be a closed proof in* NA$^\omega$*. Then there is a closed term $\{\!|\mathcal{P}|\!\}^+ : \sigma^*(C)$, such that $\lceil \{\!|\mathcal{P}|\!\}^+ \rceil \leq K(\lceil \mathcal{P} \rceil [\![\mathcal{P}]\!]^2)$, and a proof $\overline{\mathcal{P}} : \forall y^{\tau^-(C)}\ |C|_y^{(\{\!|\mathcal{P}|\!\}^+)^{\downarrow +}}$.*

Proof. Follows from Theorem 4.23 and Corollary 4.19. Note that we cannot claim the quasi-linear bound on the projected term $(\{\!|\mathcal{P}|\!\}^+)^{\downarrow +}$, since its size depends exponentially on the size of the conclusion formula C. □

Remark 4.28. The usual characterisation theorem for the original Dialectica interpretation states that in an extension of HA$^\omega$ for any formula C we are able to prove its equivalence to the formula $\exists x\, \forall y\ |C|_y^x$. Since Proposition 4.18 states syntactic equality of the formulas $|C|_y^x$ and $(\!|C|\!)_{y^{\uparrow -}}^{x^{\uparrow +}}$, the defined term mapping immediately gives us a characterisation theorem for the quasi-linear variant of the Dialectica interpretation.

4.7 Program simplification via affine reductions

Let us revisit Example 4.1 from Section 4.1. Applying directly the results from Theorem 4.23, we obtain the term

$$
\begin{aligned}
R &:= \lambda y_6\, \textbf{let } x := y_6 \textbf{ in } \lambda y_7\, \textbf{let } f_1 := s \textbf{ in } f_1 y_7, \text{ where}\\
s &:= \mathcal{R}_{\mathsf{N}}\, x\, t_0\, (\lambda x\, \lambda p\, \textbf{let } x_p := p \textbf{ in } t_1),\\
t_0 &:= \lambda y_0\, \textbf{let } y := y_0 \textbf{ in let } y_1 := y \textbf{ in } y_1,\\
t_1 &:= \lambda y_2\, \textbf{let } f_0 := (\lambda y_3\, \textbf{let } z := y_3 \textbf{ in let } y_4 := z + z \textbf{ in } y_4) \textbf{ in}\\
&\qquad\quad \textbf{let } y := y_2 \textbf{ in let } y_5 := y + 1 \textbf{ in let } z_0 := x_p y_5 \textbf{ in } f_0 z_0.
\end{aligned}
$$

This first attempt seems discouraging, because the term R is definitely larger than $[\![P]\!]^+$ from Example 4.1. However, the time complexity of R can be shown to be now linear.

One idea for simplification of R is to normalise it. However, the normal form of R is exactly $[\![P]\!]^+$, which is of exponential time complexity. To improve the readability of the quasi-linear programs, and in particular of R, another strategy for simplification needs to be chosen.

We will consider a subset of the term reduction relation, which does not increase the size of involved terms. This reduction follows Grishin's idea of logics with weakening but no contraction [Gri74, Gri81], and we refer to it as *affine*, borrowing the name for such logic as suggested by Girard. In the following, we will denote the number of free occurrences of the variable x in the term t as $\#_x(t)$.

Definition 4.29 (Affine term reductions)**.** We define the *affine term reduction* relation $\overset{a}{\mapsto}$ inductively as follows:

$$
\begin{array}{lcllllcl}
(\lambda x\, s)t & \overset{a}{\mapsto} & s\,[x := t]\,, & & \text{if } \#_x(t) \leq 1 & & & \\
\mathsf{Split}\, \langle s, t\rangle\, f & \overset{a}{\mapsto} & f\, s\, t, & & & & & \\
\mathsf{Cases}\, \mathsf{tt}\, s\, t & \overset{a}{\mapsto} & s, & & \mathcal{R}_{\mathsf{N}}\, 0\, s\, t & & \overset{a}{\mapsto} & s,\\
\mathsf{Cases}\, \mathsf{ff}\, s\, t & \overset{a}{\mapsto} & t, & & \mathcal{R}_{\mathsf{L}(\rho)}\, \mathsf{nil}\, s\, t & & \overset{a}{\mapsto} & s,
\end{array}
$$

and if $s \overset{a}{\mapsto} s'$, then

$$
sr \overset{a}{\mapsto} s'r, \qquad rs \overset{a}{\mapsto} rs', \qquad \lambda x\, s \overset{a}{\mapsto} \lambda x\, s'.
$$

The reflexive and transitive closure is denoted as usual $\overset{a*}{\mapsto}$.

Proposition 4.30. *Let $r \overset{a}{\mapsto} s$. Then*

1. $\lceil s \rceil < \lceil r \rceil$,
2. $\#_x(s) \leq \#_x(r)$.

Proof. Straightforward verification by induction on the definition of the $\overset{a}{\mapsto}$ relation. □

The reduction relation $\overset{a}{\mapsto}$ is clearly strongly normalising as a subrelation of $\mapsto$. To prove its confluence, it suffices to prove that it is *locally confluent* [BN98]. First, we prove a technical lemma.

Lemma 4.31. If $r \overset{a}{\mapsto} r'$ and $\#_x(r) = 1$, then $r\,[x := p] \overset{a*}{\mapsto} r'\,[x := p]$ for any term p;

Proof. Induction on the definition of $\overset{a}{\mapsto}$. If the base redex does not contain the variable x, then the claim is trivial. Otherwise, we need to consider only the case $(\lambda z\, s)t \overset{a}{\mapsto} s\,[z := t]$ with $\#_z(s) \leq 1$, as in all the other cases r' is a subterm of r or an application built from subterms of r and then the claim follows trivially by the definition of $\overset{a}{\mapsto}$. Since $\#_x(r) = 1$, we have that either $x \in \mathsf{FV}(s)$, or $x \in \mathsf{FV}(t)$.

Case $x \in \mathsf{FV}(s)$. Since the substitution $(\lambda z\, s)\,[x := p]$ is capture-free, we can assume that $z \notin \mathsf{FV}(p)$. Thus $\#_z(s) = \#_z(s\,[x := p])$. Then

$$(\lambda z\, s\,[x := p])t \overset{a}{\mapsto} s\,[x := p]\,[z := t] \equiv s\,[z := t]\,[x := p]\,.$$

Case $x \in \mathsf{FV}(t)$. We have

$$(\lambda z\, s)t\,[x := p] \overset{a}{\mapsto} s\,[z := t\,[x := p]] \equiv s\,[z := t]\,[x := p]\,. \qquad \square$$

Theorem 4.32 (Local confluence of affine reductions)**.** *Let $r \overset{a}{\mapsto} s_1$ and $r \overset{a}{\mapsto} s_2$. Then there is a term r', such that $s_1 \overset{a*}{\mapsto} t$ and $s_2 \overset{a*}{\mapsto} r'$.*

Proof. It is only sufficient to consider pairs of redexes, which might interfere with each other, i.e., which cannot be independently reduced in parallel. Therefore, we can restrict ourselves to the case when $r \overset{a}{\mapsto} s_1$ is in one of the forms described in the base case of the definition of the reduction relation $\overset{a}{\mapsto}$. In most of these forms s_1 is a subterm of r and the local confluence holds trivially.

Case $\mathsf{Split}\,\langle s, t\rangle\, f \overset{a}{\mapsto} f\, s\, t$. The only possibility for the reduct s_2 is $\mathsf{Split}\,\langle s', t'\rangle\, f'$, where $*'$ is a reduct of $*$ for exactly one of the terms s, t or f and is equal to $*$ for the other terms. In this case, $r' := f's't'$.

Case $(\lambda x\, s)t \overset{a}{\mapsto} s\,[x := t]$. Let us denote by s' and t' some arbitrary reducts of s and t, respectively.

Subcase $s_2 \equiv (\lambda x\, s)t'$. We set $r' := s\,[x := t']$. By induction on the term s we can prove that if $x \in \mathsf{FV}(s)$, then $s\,[x := t] \overset{a}{\mapsto} s\,[x := t']$, which is sufficient to claim that $s_1 \overset{a*}{\mapsto} r'$.

Subcase $s_2 \equiv (\lambda x\, s')t$. By Proposition 4.30, $\#_x(s') \leq 1$. Therefore, we can perform the reduction in s_2 and set $r' := s'\,[x := t]$. Finally, by Lemma 4.31 we have that $s_1 \overset{a}{\mapsto} r'$ if $x \in \mathsf{FV}(s)$, or $s_1 \equiv r'$ otherwise. $\square$

We can simplify R by considering its affine normal form. Thus we obtain

$$t_0 \overset{a}{=} \lambda y\, y, \qquad t_1 \overset{a}{=} \lambda y\, (\lambda z\, z + z)(x_p(y + 1)), \qquad \text{hence}$$

$$R \overset{a}{=} R' := \lambda x\, \mathcal{R}\, x\, (\lambda y\, y)\ (\lambda x\, \lambda p\, \lambda y\, (\lambda z\, z + z)(p(y+1)))\,.$$

Now $R'xy$ still reduces to $2^x(x+y)$ in a number of steps depending linearly on x, as opposed to its exponentially behaving counterpart $[\![P]\!]^+$ obtained by applying the original Dialectica interpretation.

4.8 Case studies revisited

In this section we subject the case studies from Chapter 3 to the new variant of the interpretation and compare the new extracted programs with the previous ones.

4.8.1 Stolzenberg's example

Table 4.1 shows the extraction process following Theorem 4.23 with some affine reductions executed. The final program $\{\!|M|\!\}$ with all affine reductions executed is shown below.

$$\{\!|M|\!\} \overset{a}{=} \lambda f \left(\begin{array}{l} \mathbf{let}\ h := \lambda\langle b, x_w\rangle \left(\begin{array}{l} \mathbf{let}\ g := \lambda k_1 \left(\begin{array}{c} \mathbf{let}\ k_{\mathsf{S}} := \mathsf{S}k_1\ \mathbf{in} \\ \langle k_{\mathsf{S}}, \langle k_1, x_w k_{\mathsf{S}}\rangle\rangle \end{array} \right) \mathbf{in} \\ \mathbf{let}\ k_0 := 0\ \mathbf{in} \\ \mathbf{let}\ z := x_w k_0\ \mathbf{in}\ \left\langle k_0 \overset{w}{\bowtie} gz_{\llcorner}, gz_{\lrcorner} \right\rangle \end{array} \right) \mathbf{in} \\ \mathbf{let}\ x_u := h_{\llcorner}\ \mathbf{in} \\ \mathbf{let}\ h_{\mathsf{tt}} := \lambda n_{\mathsf{tt}} \left(\begin{array}{l} \mathbf{let}\ h_{\mathsf{ff}} := \lambda n_{\mathsf{ff}}\ \langle n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}, n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}\rangle\ \mathbf{in} \\ \mathbf{let}\ s_{\mathsf{ff}} := \langle \mathsf{ff}, h_{\mathsf{ff}\llcorner}\rangle\ \mathbf{in}\ \langle h_{\mathsf{ff}}(x_u s_{\mathsf{ff}})_{\lrcorner}, s_{\mathsf{ff}}\rangle \end{array} \right) \mathbf{in} \\ \mathbf{let}\ s_{\mathsf{tt}} := \langle \mathsf{tt}, h_{\mathsf{tt}\llcorner}\rangle\ \mathbf{in}\ h(s_{\mathsf{tt}} \overset{u}{\bowtie} h_{\mathsf{tt}}(x_u s_{\mathsf{tt}})_{\lrcorner})_{\lrcorner} \end{array} \right)$$

The obtained program is noticeably smaller than its counterpart from Section 3.1.3. Note that there are still repeated non-trivial subterms: $n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}$. The reason is that these terms are also repeated in the proof, namely in the lemmas L_{tt} and L_{ff}.

The removal of some repetitions definitely leads to improvement, most noticeably in the Dialectica case distinctions $\overset{u}{\bowtie}$ and $\overset{w}{\bowtie}$. However, the performance gain from avoiding repetitions is only noticeable when base types are involved, since terms of type of non-zero degree are not reduced until applied to a sufficient number of arguments. Thus the **let** constructions involving h_{tt}, h_{ff}, h and g improve readability, but not evaluation speed.

$\mathcal{P}$	$\llbracket\mathcal{P}\rrbracket$	$\llbracket\mathcal{P}\rrbracket^{+}$	$\bullet$	$\llbracket\mathcal{P}\rrbracket^{-}_{\bullet}$
L_b	$\mathbf{let}\ k_b := n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}\ \mathbf{in}\ []$	ε	v_b	k_b
$L_1\ := C_B(f(n_{\mathsf{tt}} \sqcup n_{\mathsf{ff}}))$	$\llbracket L_{\mathsf{tt}}\rrbracket[\llbracket L_{\mathsf{ff}}\rrbracket]$	ε	v_b	k_b
$L_2\ := \lambda n_{\mathsf{ff}}\,\lambda v_{\mathsf{ff}}\,L_1$	$\lambda n_{\mathsf{ff}}\,\llbracket L_1\rrbracket$	k_{ff}	v_{tt}	k_{tt}
$L_3\ := \lambda n_{\mathsf{tt}}\,\lambda v_{\mathsf{tt}}\,u\,\mathsf{ff}\,L_2$	$\lambda n_{\mathsf{tt}}\,\mathbf{let}\ h_{\mathsf{ff}} := \{\!\lvert L_2\rvert\!\}\ \mathbf{in}$ $\mathbf{let}\ s_{\mathsf{ff}} := \langle \mathsf{ff}, h_{\mathsf{ff}\llcorner}\rangle\ \mathbf{in}$ $\mathbf{let}\ z_{\mathsf{ff}} := x_u s_{\mathsf{ff}}\ \mathbf{in}\ []$	$h_{\mathsf{ff}} z_{\mathsf{ff}\lrcorner}$	u	s_{ff}
$L_4\ := u\,\mathsf{tt}\,L_3$	$\mathbf{let}\ h_{\mathsf{tt}} := \{\!\lvert L_3\rvert\!\}\ \mathbf{in}$ $\mathbf{let}\ s_{\mathsf{tt}} := \langle \mathsf{tt}, h_{\mathsf{tt}\llcorner}\rangle\ \mathbf{in}$ $\mathbf{let}\ z_{\mathsf{tt}} := x_u s_{\mathsf{tt}}\ \mathbf{in}\ []$	ε	u	$s_{\mathsf{tt}} \overset{u}{\bowtie} h_{\mathsf{tt}} z_{\mathsf{tt}\lrcorner}$
$L\ \ := \lambda f\,\lambda u\,L_4$	$\lambda y\,\mathbf{let}\ f := y_{\llcorner}\ \mathbf{in}$ $\mathbf{let}\ x_u := y_{\lrcorner}\ \mathbf{in}\ \llbracket L_4\rrbracket$	$\llbracket L_4\rrbracket^{-}_{u}$		
$M_1 := v k_1 k_2 (M_{<} u_2)(M_{=} z_1 z_2)$	$\mathbf{let}\ k_{12} := \langle k_1, k_2\rangle\ \mathbf{in}\ []$	ε	v	k_{12}
$M_2 := \lambda k_2\,\lambda u_2\,\lambda z_2\,M_1$	$\lambda k_2\,\llbracket M_1\rrbracket$	ε	v	k_{12}
$M_3 := \lambda k_1\,\lambda u_1\,\lambda z_1\,w(\mathsf{S}k_1)M_2$	$\lambda k_1\,\mathbf{let}\ f := \{\!\lvert M_2\rvert\!\}\ \mathbf{in}$ $\mathbf{let}\ k_{\mathsf{S}} := \mathsf{S}k_1\ \mathbf{in}$ $\mathbf{let}\ m := x_w k_{\mathsf{S}}\ \mathbf{in}\ []$	ε	v w	fm k_{S}
$M_4 := w\,0\,M_3$	$\mathbf{let}\ g := \{\!\lvert M_3\rvert\!\}\ \mathbf{in}$ $\mathbf{let}\ k_0 := 0\ \mathbf{in}$ $\mathbf{let}\ z := x_w k_0\ \mathbf{in}\ []$	ε	v w	$g z_{\llcorner}$ $k_0 \overset{w}{\bowtie} g z_{\lrcorner}$
$M_5 := \lambda b\,\lambda w\,M_4$	$\lambda s\,\mathbf{let}\ b := s_{\llcorner}\ \mathbf{in}$ $\mathbf{let}\ x_w := s_{\lrcorner}\ \mathbf{in}\ \llbracket M_4\rrbracket$	$\llbracket M_4\rrbracket^{-}_{w}$	v	$\llbracket M_4\rrbracket^{-}_{v}$
$M\ \ := \lambda f\,\lambda v\,L f M_5$	$\lambda f\,\mathbf{let}\ h := \{\!\lvert M_5\rvert\!\}\ \mathbf{in}$ $\llbracket L\rrbracket\,\langle f, h_{\llcorner}\rangle\,[\mathbf{let}\ s := \llbracket L\rrbracket^{+}\ \mathbf{in}\ []]$	$h s_{\lrcorner}$		

Table 4.1: Quasi-linear extraction from Stolzenberg's example

4.8.2 Infinite Pigeonhole Principle

Applying the results from Theorem 4.23 on hand can become cumbersome when the complexity of the proof increases. This can be seen in the extraction from the Infinite Pigeonhole Principle case study, which is carried out thoroughly in Tables 4.2 and 4.3. Even though the obtained program is noticeably smaller, the extraction process itself is more complicated, because of the many variables involved. We postpone the unfolding of the program $\{\!|M|\!\}$ until the end of next chapter, where the unnecessary computations will be removed and the whole program will be more readable.

Let us estimate the worst time complexity of $\{\!|M|\!\}$ in a similar fashion to Section 3.3.4. Applying $\{\!|M|\!\}$ to 0 colours invokes $\{\!|L|\!\}$ once with appropriate arguments and produce a pair of results $\langle q, x_w \rangle$. Then x_w is used to construct a list of length n and to compute a counterexample index, both of which are done simultaneously in n steps.

Now let us assume that $\{\!|M|\!\}$ executes $\{\!|L|\!\}$ for $r+1$ colours. We will estimate the number of recursive reductions, which lead to recursion with r colours. The recursive call to x_p occurs in $\{\!|L_3|\!\}$, which is being invoked by g_3. On the other hand, g_3 appears twice: applied to z_6 to compute z_7 and as a function in z_5, which is used an argument of x_{u_2}. The first occurrence of g_3 leads to one recursive call when z_7 needs to be reduced, but the second one depends on the number of invocations of the parameter x_w in $\{\!|M_8|\!\}$, which is essentially the value of x_{u_2}. We should note that x_{u_2} is invoked two different times with z_5: once to compute z_6 and once to calculate the case distinction in $[\![L_4]\!]^-_{u_2}$. When we trace the use of x_w in $\{\!|M_8|\!\}$, we have a similar situation, namely that x_w is used twice for every number n: once to calculate the next index in the list z_4 by applying z_3 to the previous one and once in the case distinction $\overset{w}{\bowtie}$.

In total we have $4n$ recursive reductions at each recursive step for r, hence the worst time complexity of $\{\!|M|\!\}$ becomes $O((4n)^r)$. Although the complexity is still exponential, it is a clear improvement from the factorial worst and average time complexity $O(r!(3n)^r)$, which was established in Section 3.3.4. By the same reasoning as before, the average time complexity is not better than the worst case, since all case distinctions are always evaluated and hence the recursions are fully executed. We will revisit the Infinite Pigeonhole Principle example in Chapter 6, where we will show how the average time complexity can be improved.

$\mathcal{P}$	$\llbracket\mathcal{P}\rrbracket$	$\llbracket\mathcal{P}\rrbracket^{+}$	$\bullet$	$\llbracket\mathcal{P}\rrbracket^{-}_{\bullet}$
K_1	$\lambda n_2\,[]$	ε	u_1, v_1	$n_1 \sqcup n_2$
$L_1 := \lambda m\,\lambda a\,v_2(n_1 \sqcup m)(L_{\leq} a)$	$\lambda m\,[]$	ε	v_2	$n_1 \sqcup m$
$L_2 := \lambda n_2\,\lambda v_2\,w n_2 L_1$	$\lambda n_2\,$**let** $z_1 := \{\!\vert L_1\vert\!\}$ **in** **let** $m_2 := x_w n_2$ **in** $[]$	$z_1 m_2$	w	n_2
$K_2 := \lambda q\,\lambda w\,u_2 q L_2$	$\lambda\langle q, x_w\rangle$ **let** $z_2 := \left\langle q, \{\!\vert L_2\vert\!\}^{+}\right\rangle$ **in** $[]$	$x_{u_2} z_2$	u_2	z_2
$L_3 := p(f\lceil n_1)K_1 K_2$	**let** $g_1 := \{\!\vert K_1\vert\!\}$ **in** **let** $g_2 := \{\!\vert K_2\vert\!\}$ **in** **let** $z_3 := \langle f\lceil n_1, g_{2\llcorner}\rangle$ **in** **let** $z_4 := x_p z_3$ **in** $[]$	ε	u_1, v_1 u_2 p	$g_1(z_{4\lrcorner}) \triangleright \bullet$ $g_2(z_{4\llcorner})_{\lrcorner}$ z_3
$L_4 := u_2 r(\lambda n_1\,\lambda v_1\,L_3)$	**let** $g_3 := \lambda n_1\,\{\!\vert L_3\vert\!\}$ **in** **let** $z_5 := \langle r, g_3 \triangleright v_1\rangle$ **in** **let** $z_6 := x_{u_2} z_5$ **in** **let** $z_7 := g_3 z_6$ **in** $[]$	ε	u_1, p u_2	$z_7 \triangleright \bullet$ $z_5 \overset{u_2}{\bowtie} (z_7 \triangleright u_2)$
$L_{\mathsf{S}} := \lambda f\,\lambda u_1\,\lambda u_2\,L_4$	$\lambda y\,$**let** $f := y_{\llcorner}$ **in** **let** $x_{u_2} := y_{\lrcorner}$ **in** $\llbracket L_4\rrbracket$	$\left\langle \llbracket L_4\rrbracket^{-}_{u_2}, \llbracket L_4\rrbracket^{-}_{u_1}\right\rangle$	p	$\llbracket L_4\rrbracket^{-}_{p}$
L_0	$\lambda y_0\,[]$	$\square^{\mathsf{N}\times(\mathsf{N}\Rightarrow\mathsf{N})\times\mathsf{N}}$		
L	$\lambda\langle r, y\rangle\,\mathcal{R}_{\mathsf{N}} r\{\!\vert L_0\vert\!\}(\lambda r\,\lambda x_p\,\{\!\vert L_{\mathsf{S}}\vert\!\})y$			

Table 4.2: Quasi-linear extraction from the Infinite Pigeonhole Principle

$\mathcal{P}$	$\llbracket\mathcal{P}\rrbracket$	$\llbracket\mathcal{P}\rrbracket^+$	$\bullet$	$\llbracket\mathcal{P}\rrbracket^-_\bullet$
$M_{\mathsf{S}}^{(i)}$	$\lambda k\, \mathcal{R}_{\mathsf{N}} k\, \square^{\mathsf{N}}\, (\lambda k\, \lambda p\, [])$	ε	v_i	$p \overset{v_i}{\bowtie} k$
$M_3 := \lambda m\, \lambda w_1\, \lambda w_2\, u_{\mathsf{S}n}(m :: x :: l) v_0 M'_{\mathsf{S}} M''_{\mathsf{S}}$	$\lambda m\, \mathbf{let}\ z_1 := m :: x :: l\ \mathbf{in}$ $\mathbf{let}\ z_2 := x_{u_{\mathsf{S}n}} z_1\ \mathbf{in}$ $\mathbf{let}\ h_i := \{\!\vert M_{\mathsf{S}}^{(i)} \vert\!\} z_2\ \mathbf{in}\ []$	ε	$u_{\mathsf{S}n}$ v_i	z_1 h_i
$M_4 := \lambda v_0\, \lambda v_1\, \lambda v_2\, w(\mathsf{S}x) M_3$	$\mathbf{let}\ z_3 := \mathsf{S}x\ \mathbf{in}$ $\mathbf{let}\ z_4 := \{\!\vert M_3 \vert\!\}(x_w z_3)\ \mathbf{in}\ []$	$\langle z_4 \triangleright v_1,$ $z_4 \triangleright v_2\rangle$	$u_{\mathsf{S}n}$ w	$z_4 \triangleright u_{\mathsf{S}n}$ z_3
$M_5 := \lambda l\, \mathsf{Ind}\, l\, \mathsf{efq}(\lambda x\, \lambda l\, \lambda p'\, M_4)$	$\lambda l\, \mathbf{let}\ z_5 := \mathcal{R}_{\mathsf{L(N)}} l\, \square\, (\lambda x\, \lambda l\, \lambda x_{p'}\, \{\!\vert M_4 \vert\!\})\ \mathbf{in}\ []$	$z_{5\llcorner}$	$u_{\mathsf{S}n}, w$	$z_5 \triangleright \bullet$
$M_{\mathsf{S}} := \lambda u_{\mathsf{S}n}\, p M_5$	$\lambda x_{\mathsf{S}n}\, \mathbf{let}\ h_4 := \{\!\vert M_5 \vert\!\}\ \mathbf{in}$ $\mathbf{let}\ h_5 := h_{4\llcorner}\ \mathbf{in}$ $\mathbf{let}\ z_6 := h_4(x_p h_5)\ \mathbf{in}\ []$	$z_6 \triangleright u_{\mathsf{S}n}$	w p	$z_6 \triangleright w$ h_5
$M_6 :=\ \lambda m\, \lambda w_1\, \lambda w_2\, u_0 (m{:}) \mathsf{AxT}(\lambda k\, \mathsf{efq})$ $(\lambda k\, w_2)$	$\lambda m\, []$	ε	u_0	$m{:}$
$M_0 := \lambda u_0\, w 0 M_6$	$\lambda x_0\, \mathbf{let}\ z_7 := 0\ \mathbf{in}$ $\mathbf{let}\ z_8 := \{\!\vert M_6 \vert\!\}(x_w z_7)\ \mathbf{in}\ []$	z_8	w	z_7
$M_1 := \mathsf{Ind}\, n\, M_0\, (\lambda n\, \lambda p\, M_{\mathsf{S}})$	$\lambda y\, \mathbf{let}\ z_9 := \mathcal{R}_{\mathsf{N}}\, n\, \{\!\vert M_0 \vert\!\} \big(\lambda n\, \lambda p\, \mathbf{let}\ x_p := p_\llcorner\ \mathbf{in}$ $\llbracket M_{\mathsf{S}} \rrbracket [\Big\langle \llbracket M_{\mathsf{S}} \rrbracket^+, p h_{5\lrcorner} \overset{w}{\bowtie} \llbracket M_{\mathsf{S}} \rrbracket^-_w \Big\rangle]\big) y\ \mathbf{in}\ []$	$z_{9\llcorner}$	w	$z_{9\lrcorner}$
M_2	$\lambda\langle q, k\rangle\, \mathbf{let}\ k' := \mathsf{S}k\ \mathbf{in}\ []$	$k \overset{u}{\bowtie} k'$		
$M_7 := \lambda l\, \lambda v_0\, \lambda v_1\, \lambda v_2\, v l v_0 v_1 (M_2 q v_2)$	$\lambda l\, \mathbf{let}\ z_{10} := x_v l\ \mathbf{in}$ $\mathbf{let}\ z_{11} := \{\!\vert M_2 \vert\!\}\, \langle q, z_{10\lrcorner}\rangle\ \mathbf{in}\ []$	$\langle z_{10\llcorner}, z_{11}\rangle$	v	l
$M_8 := \lambda q\, \lambda w\, M_1 M_7$	$\lambda\langle q, x_w\rangle\, \mathbf{let}\ z_{12} := \{\!\vert M_7 \vert\!\}\ \mathbf{in}$ $\llbracket M_1 \rrbracket (z_{12\llcorner}) [\mathbf{let}\ z_{13} := \llbracket M_1 \rrbracket^+\ \mathbf{in}\ []]$	$\llbracket M_1 \rrbracket^-_w$	v	$z_{12\lrcorner} z_{13}$
$M\ := \lambda r\, \lambda f\, \lambda z\, \lambda n\, \lambda v\, L r f z M_8$	$\lambda\langle r, f, n, x_v\rangle\, \mathbf{let}\ z_{14} := \{\!\vert M_8 \vert\!\}\ \mathbf{in}$ $\mathbf{let}\ z_{15} := \{\!\vert L \vert\!\}\, \langle r, f, z_{14\llcorner}\rangle\ \mathbf{in}\ []$	$\langle z_{15\lrcorner}, z_{14\lrcorner}(z_{15\llcorner})\rangle$		

Table 4.3: Quasi-linear extraction from Unbounded Pigeonhole Principle

CHAPTER

FIVE

DIALECTICA INTERPRETATION WITH FINE COMPUTATIONAL CONTROL

Computational interpretations, such as modified realisability or Dialectica, aim at extracting the maximal amount of algorithmic information from a proof, regardless if this information is required or redundant. One example of such an irrelevant computation is a function, which produces the same result when applied to every possible value of its argument type. Finding such semantical dependencies in general is not an easy task, however a simple syntactic criterion for detecting some of these redundancies is to search for parts of the program which have superfluous parameters, i.e., terms of the form $\lambda x\, r$ with $x \notin \mathsf{FV}(r)$. If such a function is applied to any other term t, then the result of evaluating t will be lost, so t is redundant subterm and its evaluation is unnecessary.

In [Ber05] Berger showed that in modified realisability such redundancies in the extracted program can appear due to introductions of $\forall x$, such that the variable x appears only in terms which are not used in a computational manner. Non-computational variants of quantifiers with only logical meaning, i.e., *computationally uniform quantifiers*, were shown to be interpretable and their use in the proofs leads to discarding redundant parameters in the extracted programs. A concrete example in [Ber05] of a proof of totality of the list reversal function demonstrated that removal of redundant parameters can decrease time complexity of the extracted program under strict evaluation.

Following this idea, Hernest transferred the concept of uniform quantifiers to the Dialectica interpretation [Her07a, Her07b]. Because the Dialectica interpretation extracts more computational information than modified realisability, the need for using uniform quantifiers becomes even stronger. In particular, they can be used to avoid redundant case distinctions. On the other hand, the use of uniform quantifiers can lead to an undecidable translation, which imposes additional restrictions. However, a recent joint work by Hernest and the author showed that the dual nature of the Dialectica interpretation allows for two independent sorts of computational uniformities for the universal quantifier [HT10]. This result was further extended to implication, allowing for a vast variety of combinations and a very fine level of computational control [Tri09].

In this chapter a fine computational version of the quasi-linear interpretation will be presented. It will be shown how the uniform quantifiers can be used to optimise the extracted programs from Chapter 3. Moreover, the finer uniform connectives allow modelling modified realisability inside the Dialectica interpretation in a similar way as suggested by Hernest and Oliva in [HO08]. The results in the present chapter are mainly based on [Tri09], but are adjusted to the interpretation from Chapter 4, which allows for a more systematic treatment of the uniform annotations of implication.

5.1 Examples of redundant computation

The main motivation for introducing the uniform quantifiers was to remove redundant parameters in programs extracted via modified realisability. This need can be illustrated by the following simple example, which is an adaptation of an example given by Monika Seisenberger.

Example 5.1. Let us consider the simple statement that the square of every even number must be divisible by four. Formally,

$$A := \forall n \, \forall m \, \big(n = 2m \to \exists k \, (n^2 = 4k)\big)$$

There are two essentially different constructive proofs of this statement, namely

$$\begin{aligned} M_1 &:= \lambda n \, \lambda m \, \lambda u^{n=2m} \, \big\langle m^2, L_1 \big\rangle \, , \\ M_2 &:= \lambda n \, \lambda m \, \lambda u^{n=2m} \, \big\langle n^2/4, L_2 \big\rangle \, , \end{aligned}$$

where L_1 and L_2 are the necessary equality lemmas and a/b denotes integer division. Applying modified realisability we obtain the following two programs

$$\begin{aligned} [\![M_1]\!]^\circ &\equiv \lambda n \, \lambda m \, m^2, \\ [\![M_2]\!]^\circ &\equiv \lambda n \, \lambda m \, (n^2/4). \end{aligned}$$

Obviously, both programs have a redundant parameter: n in $[\![M_1]\!]^\circ$ and m in $[\![M_2]\!]^\circ$. This superfluity may not seem very harmful if we consider the statement out of context. However, the situation is more different when the proofs M_i are used as lemmas in a larger proof. Then the corresponding programs $[\![M_i]\!]^\circ$ will be applied to some terms t_n and t_m computing n and its half m respectively, while in fact only one of them is needed depending on which of the proofs was used.

The reason for this phenomenon is that the parameters n and m are connected by the equality $n = 2m$ and thus the value of one of them is determined by the value of the other. One way to fix the redundancy is to omit one of the parameters by rephrasing the statement, for example as

$$B := \forall n \left(2(n/2) = n \to \exists k\,(n^2 = 4k)\right).$$

However, this approach is not desirable for at least two reasons:

1. the modification needs to be done by hand,
2. the modification induces a respective possibly non-trivial change in the proofs which use the statement as a lemma.

Instead, we would like to have an instrument, which allows to insert annotations in the statement in a way to denote lack of computational meaning. This can be done by using a uniform quantifier $\forall^{\mathsf{U}}$. We impose the restriction that the proof $\lambda x\, M : \forall^{\mathsf{U}} x\, A$ is valid only if x does not appear in M computationally, i.e., $x \notin \mathsf{FV}(\llbracket M \rrbracket^\circ)$. Then we have two possibilities for annotating the formula A:

$$A_1 := \forall^{\mathsf{U}} n\, \forall m \left(n = 2m \to \exists k\,(n^2 = 4k)\right),$$
$$A_2 := \forall n\, \forall^{\mathsf{U}} m \left(n = 2m \to \exists k\,(n^2 = 4k)\right).$$

M_i is a valid proof for A_i, but not for A_{3-i}. Moreover, the proof M_i of the original statement A determines uniquely a "maximal" annotation A_i, which removes all possible redundancies, without removing additional content.

Note that A can be equivalently formulated as

$$C := \forall n \left(\exists m\,(n = 2m) \to \exists k\,(n^2 = 4k)\right).$$

This form of the statement has two similar proofs

$$N_1 := \lambda n\, \lambda v^{\exists m\,(n=2m)}\, \exists^- v(\lambda m\, \lambda u^{n=2m}\, \langle m^2, K_1 \rangle),$$
$$N_2 := \lambda n\, \lambda v^{\exists m\,(n=2m)}\, \exists^- v(\lambda m\, \lambda u^{n=2m}\, \langle n^2/4, K_2 \rangle),$$

from which we can extract the programs

$$\llbracket N_1 \rrbracket^\circ = \lambda n\, \lambda m\, (\lambda x\, \lambda f\, f x) m (\lambda m\, \langle m^2, K_1 \rangle),$$
$$\llbracket N_2 \rrbracket^\circ = \lambda n\, \lambda m\, (\lambda x\, \lambda f\, f x) m (\lambda m\, \langle n^2/4, K_2 \rangle)).$$

It is easy to see that $\llbracket M_i \rrbracket^\circ$ are the normal forms of the programs $\llbracket N_i \rrbracket^\circ$ and thus the latter suffer from the same deficiencies as the former. In order to repair the problem for $[\![N_2]\!]^\circ$, we would need to use the uniform version of the existential quantifier $\exists^{\mathsf{U}}$. Similarly to its universal counterpart, it signifies computational irrelevance of the claimed witness. The respective restriction is on the elimination rule, which allows elimination $\exists^- M^{\exists^{\mathsf{U}} x\, A}(\lambda x\, \lambda u^A\, N)$ only if x is not used computationally in N, i.e., $x \notin \mathsf{FV}([\![N]\!]^\circ)$.

A natural question to consider is whether such optimisations are possible when extracting from proofs which use non-constructive principles. Berger already showed in his paper [Ber05] how uniform quantifiers can improve programs obtained from a proof from contradiction. This was possible, because the extraction method was refined A-translation, which is essentially based on modified realisability. The first adaptation of the uniform quantifiers to the Dialectica interpretation was given by Hernest [Her07b] and was called "Light Dialectica interpretation". Hernest noticed that there is a substantial difference in the uniformity restrictions, compared to the simpler case of modified realisability. Dialectica collects not one, but two orthogonal pieces of computational information — witnesses and counterexamples, where the witnesses are functions taking counterexample candidates as parameters. The main problem arises from the requirement for a quantifier-free Dialectica translation $|A|^x_y$.

Consider an assumption $\forall^{\mathsf{U}} x\, A$ with $\forall^{\mathsf{U}} x$ denoting a uniform universal quantifier. Its Dialectica translation is $\left|\forall^{\mathsf{U}} x\, A\right|^r_u := \forall x\ |A|^r_u$, which means that the witness r:

(+) does not depend on x, but

(−) needs to be valid for all possible values for x

However, if the assumption $\forall^{\mathsf{U}} x\, A$ is used more than once, then we will not be able to decide which of the extracted counterexamples for u should be used for r. On the other hand, it might turn out that r solves the formula without using *any* counterexamples of A. In this case we could convert all positively occurring universal quantifiers in A to uniform and we will be able to produce a more efficient program, which does not produce any counterexamples for A at all, and hence requires no case distinctions. The described situation leads to the following restriction, defined by Hernest in [Her07b], which is unique to the Dialectica interpretation:

> If an assumption A is used more than once and requires counterexamples, its formula cannot contain the universal uniform quantifier $\forall^{\mathsf{U}}$.

Because of the dual nature of the Dialectica interpretation, it turns out that more refined variants of the uniform quantifiers can be considered. Hernest and the author observed that the conditions (+) and (−) above can be imposed separately and independently for each quantifier occurrence. This gives rise to four different sorts of universal quantifiers, considered in [HT10]: one fully computational, one fully uniform, and two semi-uniform quantifiers: only positively and only negatively uniform, respectively. The idea for the semi-uniform quantifiers was inspired by the following motivating example due to Paulo Oliva.

Example 5.2. Consider a predicate P on natural numbers and the statement that if P holds for infinitely many natural numbers, then P holds for numbers which are arbitrarily apart. Formally,

$$\forall x\, \tilde{\exists} y\, (y > x \mathbin{\tilde{\wedge}} P(y)) \to \forall d\, \tilde{\exists} n_1, n_2\, (n_2 > n_1 + d \mathbin{\tilde{\wedge}} P(n_1) \mathbin{\tilde{\wedge}} P(n_2)).$$

A proof of the statement needs to use the assumption twice, once for an arbitrary x, say 0, to obtain n_1 and then once more for $x := n_1 + d$ to obtain n_2:

$$\begin{aligned} M &:= \lambda u\, \lambda d\, \lambda v\, u0 \qquad (\lambda n_1\, \lambda w_1^{n_1>0} \quad \lambda z_2^{P(n_1)} \\ &\qquad\qquad u(n_1+d)(\lambda n_2\, \lambda w_2^{n_2>n_1+d}\, \lambda z_2^{P(n_2)}\, v n_1 n_2 w_2 z_1 z_2)), \text{ where} \\ u &: \forall x\, \tilde{\exists} y\, (y > x \mathbin{\tilde{\wedge}} P(y)), \\ v &: \forall n_1\, \forall n_2\, (n_2 > n_1 + d \to P(n_1) \to P(n_2) \to \mathrm{F}). \end{aligned}$$

The program extracted from M using the quasi-linear Dialectica interpretation is

$$\begin{aligned} \{\!|M|\!\} \overset{a}{=} \lambda f\, \lambda d\, &\textbf{let } x_1 := 0 \textbf{ in let } n_1 := f x_1 \textbf{ in} \\ &\qquad \textbf{let } x_2 := n_1 + d \textbf{ in let } n_2 := f x_2 \textbf{ in } \left\langle \langle n_1, n_2 \rangle, x_1 \overset{u}{\bowtie} x_2 \right\rangle, \text{ where} \\ x_1 \overset{u}{\bowtie} x_2 \equiv &\ \mathsf{Cases}\, (n_1 > x_1 \wedge (\!|P(n_1)|\!)^{\varepsilon}_{\varepsilon})^{\mathrm{at}}\, x_2\, x_1. \end{aligned}$$

The term $\{\!|M|\!\}$ contains simultaneously positive and negative information about the proof: on one hand, it computes the witnesses for $\tilde{\exists} n_1, n_2$ and on the other it computes a single counterexample for $\forall x$, using a case distinction on the Dialectica translation of the predicate P.

In case M is a part of a larger proof, similarly to the Infinite Pigeonhole Principle in Section 3.3, then we could need the counterexample index for x in order to obtain a backtracking effect. However, it might happen that M is used in a context, where only the positive witness pair $\langle n_1, n_2 \rangle$ is needed. Then computing the case distinction $x_1 \overset{u}{\bowtie} x_2$ is redundant and might be computationally expensive, depending on the form of P. Moreover, if P is an undecidable predicate, then we will not be able to interpret the statement, which should not be the case if we want to obtain only the positive witnesses, which do not depend on the form of P.

The most obvious idea to separate only the positive half of $\{\!|M|\!\}$ is to disable the computational content of x by marking $\forall^{\mathsf{U}} x$. However, this would mean that y does not computationally on x, i.e., it is a constant. This is clearly impossible, as there is no natural number larger than every natural number (including itself). Even though that there is no violation of the uniformity constraints, M would be unusable as there would be no way to prove its premise.

The example demonstrates the dual nature of Dialectica witnesses. On one hand, x is a parameter of the function f (positive meaning), and on the other, it might be a counterexample to the correctness of the function f (negative meaning). In this case we would like to discard the negative meaning of x while keeping its positive meaning. This can be achieved by introducing two independent uniformities, as suggested in [HT10], each corresponding to separately requiring one of $(+)$ and $(-)$ above. The solution to this example is to require only $(-)$ and not $(+)$, i.e., to use a negatively uniform, yet positively computational quantifier. Then we would obtain the following program:

$$\lambda f\, \lambda d\, \mathbf{let}\ n_1 := f0\ \mathbf{in}\ \mathbf{let}\ n_2 := f(n_1 + d)\ \mathbf{in}\ \langle n_1, n_2\rangle\,.$$

It is worth noting that by discarding the counterexample for x in the example above, we effectively obtain the same content, which we would have obtained by applying modified realisability to a constructive formulation of the proof:

$$M' := \lambda u\, \lambda d\, \exists^-(u0)(\lambda n_1\, \lambda w_1\, \lambda z_2\, \exists^-(u(n_1+d))(\lambda n_2\, \lambda w_2\, \lambda z_2\, \langle n_1, n_2, w_2, z_1, z_2\rangle)).$$

We should also note that the semi-uniform quantifier is only meaningful if the positive computational content of the quantified formula is non-trivial, otherwise it would not make sense to take special care to preserve the anyway void positive dependency of the quantified variable.

An interesting question to consider is whether we can use the Dialectica interpretation to mimic the behaviour of modified realisability on negatively formulated but essentially constructive proofs, by applying the new semi-uniform annotations. It turns out that the annotations on the uniform quantifiers are not sufficient, as can be demonstrated by an example, similar to Example 5.2.

Example 5.3. Let $Q(n, m)$ be a binary predicate on N and consider the formula

$$\forall m\, (\tilde{\exists} n\, Q(n, m) \to \tilde{\exists} n\, Q(n, \mathsf{S}m)) \to \tilde{\exists} n\, Q(n, 0) \to \tilde{\exists} n\, Q(n, 2).$$

The obvious proof of this statement uses the premise twice:

$$M := \lambda u^{\forall m\, (\tilde{\exists} n\, Q(n,m) \to \tilde{\exists} n\, Q(n,\mathsf{S}m))}\, \lambda v_0^{\tilde{\exists} n\, Q(n,0)}\, u1(u0v_0).$$

The extracted program is presented below:

$$\{\!|M|\!\} \overset{a}{=} \lambda f^{\mathsf{N}\Rightarrow\mathsf{N}\Rightarrow\mathsf{N}}\, \lambda n_0 \left(\begin{array}{l} \mathbf{let}\ m_0 := 0\ \mathbf{in}\ \mathbf{let}\ n_1 := f m_0 n_0\ \mathbf{in} \\ \mathbf{let}\ m_1 := 1\ \mathbf{in}\ \mathbf{let}\ n_2 := f m_1 n_1\ \mathbf{in} \\ \left\langle n_2, \langle m_0, n_0\rangle \overset{u}{\bowtie} \langle m_1, n_1\rangle \right\rangle \end{array} \right)$$

We are faced with the same situation as in Example 5.2: the program $\{\!|M|\!\}$ computes a complicated counterexample, which might not be needed. However, in this case the negative content of the assumption u is formed by two components: a counterexample for m and a counterexample for n such that $Q(n, m)$. An attempt to mark both quantifiers $\forall n$ and $\tilde{\exists} m$ as computationally irrelevant would reduce the function f to a constant, implying that in fact $\tilde{\exists} n\, \forall m\, Q(n, m)$, which clearly reduces the generality of the formula. Hence, we should aim at applying a finer uniform annotation.

Since $\tilde{\exists} n\, Q(n, m) = \neg\forall n\, \neg Q(n, m)$ and $\neg Q(n, m)$ does not require witnesses, the semi-uniform quantifier has the same effect as the ordinary fully uniform quantifier for n. On the other hand, a semi-uniform annotation only for $\forall m$ is not sufficient, because the negative computational content of u would still be non-trivial, due to the presence of positive content of the antecedent. Moreover, the situation is actually worse: the application of the semi-uniform annotation to $\forall n$ is not sound anymore, as it would introduce a universal quantifier in the Dialectica translation of u, while still requiring challenges, thus making it impossible to perform a case distinction over it.

The only solution seems to involve defining uniform annotations on implication that control the quantity of negative computational content being generated by its antecedent and by its consequent. In this case we would like to discard the negative computational contribution of the antecedent $\tilde{\exists} n\, Q(n, m)$, i.e., the generated counterexample for n, while keeping its positive computational contribution so that the function f remains binary, depending on both m and n. Thus, by applying refined uniform annotations to both the universal quantifier and the implication we obtain the simplified content

$$\lambda f^{\mathsf{N} \Rightarrow \mathsf{N} \Rightarrow \mathsf{N}}\, \lambda n_0\, f1(f0n_0).$$

In fact, this is the pure positive content of the proof M, which would be obtained by applying modified realisability to the constructive alternative:

$$M' := \lambda u^{\forall m\, (\exists n\, Q(n,m) \to \exists n\, Q(n, \mathsf{S}m))}\, \lambda v_0^{\exists n\, Q(n,0)}\, u1(u0v_0).$$

Note that due to the duality of computational content in the Dialectica interpretation, we might also need to discard the negative contribution of the *consequent* of the implication. This can be seen if the example above is reformulated as follows:

$$\forall m\, (\forall n\, \neg Q(n, \mathsf{S}m) \to \forall n\, \neg Q(n, m)) \to \tilde{\exists} n\, Q(n, 0) \to \tilde{\exists} n\, Q(n, 2).$$

The difference here is that when using the premise, the dependency between the negative contents is what matters computationally. We are faced with the same problem as above. In order to resolve it, we need to keep the positive contribution of the negative content of $\forall n \neg Q(n, m)$, but to disable its negative meaning, so that no counterexample is extracted.

A similar example can be constructed, where we need to combine both types of semi-uniform application in order to achieve the desired effect.

In the following sections we will define and examine in more detail the uniform annotations that allow for very fine computational control over the extracted programs. We will also demonstrate how the new annotations will be sufficient for modelling modified realisability in the context of the Dialectica interpretation.

There are other approaches for restricting the computational content in the Dialectica interpretation and in particular for simulating modified realisability by extracting only positive computational content. Oliva has shown that substructural logics, such as linear logic, present a suitable framework for unifying various functional interpretations [Oli08]. Expanding on the idea, Oliva and Hernest showed that different interpretations of the linear modalities can soundly coexist within the same proof, roughly corresponding to local decisions whether to discard certain parts of its computational content [HO08]. Thus a hybrid interpretation is obtained, one extreme of which is the original Dialectica interpretation, and the other is modified realisability. Later in the chapter we will discuss some relations between the different approaches for controlling computational content.

5.2 Notions of uniformity for the quasi-linear Dialectica interpretation

The uniform annotations of Hernest [Her07b, Her07a] as well as their refinements [HT10, Tri09] were presented in terms of the original Dialectica interpretation. A simple intuitive explanation behind the idea for semi-uniform quantifiers can be given using the positive and negative computational types $\tau^+(A)$ and $\tau^-(A)$. Every uniform flag corresponds to removing a component of the computational type: an *argument type* in the case of a positively uniform flag or a *factor of a product type* in the case of a negatively uniform one. The fine uniform annotations for the original Dialectica interpretation and the respective discarded type components are displayed in Figure 5.1, using the notation from [Tri09].

In the case of the quasi-linear Dialectica interpretation presented in Chapter 4, the duality between computational types is made extremely explicit. Namely, the negative computational type $\sigma^-(A)$ is directly used as the (only) parameter to the common definition context of type $\sigma^-(A) \Rightarrow \diamond$. As a result, the achieved factoring of types, which is favourable for controlling the size of the extracted term, has a restrictive effect on uniform annotations. Since we use a definition context to merge

$$\tau^+(\forall x\, A) = \underbrace{\rho}_{\overset{+}{\forall}} \Rightarrow \tau^+(A) \qquad \tau^-(\forall x\, A) = \underbrace{\rho}_{\overset{-}{\forall}} \times \tau^-(A)$$

$$\tau^+(A \to B) = (\underbrace{\tau^+(A)}_{\overset{\#}{\to}} \Rightarrow \tau^+(B)) \times (\underbrace{\tau^+(A)}_{\overset{\pm}{\to}} \Rightarrow \underbrace{\tau^-(B)}_{\overset{=}{\to}} \Rightarrow \tau^-(A))$$

$$\tau^-(A \to B) = \underbrace{\tau^+(A)}_{\underset{+}{\to}} \times \underbrace{\tau^-(B)}_{\underset{-}{\to}}$$

Figure 5.1: Uniformity annotations for the original Dialectica interpretation

common parameters, their separate roles in the computational content cannot be so easily discerned, as opposed to the original variant of the interpretation. Therefore, we have less possibilities for expressing computational uniformities.

Due to the tight connection between positive and negative content in the quasi-linear variant of the interpretation, if we disable the negative computational meaning of a component in $\sigma^-(A)$, we automatically disable also its positive computational meaning in $\sigma^*(A) = \sigma^-(A) \Rightarrow \sigma^+(A)$. In fact, we disable the respective component in the whole definition context of type $\sigma^-(A) \Rightarrow \diamond$. In the case of $A = \forall x^\rho\, B$, discarding ρ from $\sigma^-(A)$ corresponds exactly to the full uniform quantifier considered by Hernest [Her07b]. If $A = B \to C$, we can similarly discard $\sigma^*(B)$ from $\sigma^-(A)$, which can be seen as a *fully uniform implication*, similar to the one defined by Ratiu and Schwichtenberg for modified realisability [RS09]. Hence, fully uniform connectives have a very natural and easy representation in the new variant of the interpretation.

The situation with semi-uniform annotations is more complicated. When positive and negative extracted terms are defined simultaneously, it does not seem possible to consider the positive and negative contributions of a certain component independently of each other. However, in Section 4.1 we showed that semi-uniform quantifiers are very meaningful in certain cases, even when the quasi-linear variant of the interpretation is being used. In order to clarify the role of semi-uniform annotations, let us consider the three possible uniform annotations on the universal quantifier, specified in Table 5.1.

$\overset{+}{\forall}$	positively uniform	$(\!\lvert \overset{+}{\forall} x\, A \rvert\!)^r_s = (\!\lvert A\,[x := s_\llcorner]\, \rvert\!)^r_{s_\lrcorner}$
$\overset{-}{\forall}$	negatively uniform	$(\!\lvert \overset{-}{\forall} x\, A \rvert\!)^r_s = \forall x\, (\!\lvert A \rvert\!)^{r \circ x}_s$
$\overset{\pm}{\forall}$	fully uniform	$(\!\lvert \overset{\pm}{\forall} x\, A \rvert\!)^r_s = \forall x\, (\!\lvert A \rvert\!)^r_s$

Table 5.1: Uniform annotations for $\forall x\, A$

$\overset{+}{\forall}$ would be used in case we would like to remove the computational dependency of x on the positive content r of A, but still allow to specify a counterexample of it, namely s_{L}. However, when interpreting an $\overset{+}{\forall}$-introduction in the general case where we have open assumptions C_i, the computed challenges $\{\!|M|\!\}_i^-$ can still depend on x. Since the positive content of $\overset{+}{\forall} x\, A$ and the negative content of the assumptions are computed simultaneously by $\{\!|M|\!\}$, the fact that the component $\{\!|M|\!\}^+$ does not contain x is of little importance. Indeed, a challenge for x needs to be anyway applied to the common context $\llbracket M \rrbracket$ in order to compute $\{\!|M|\!\}_i^-$, which depend on x, even if $\llbracket M \rrbracket^+$ does not. By this argument, it is obvious that in the setting of common contexts positive uniformity plays no special role and thus will be not be considered. We should note that in the original interpretation the positively uniform flags also have a minor cleaning effect and are thus termed "weak" in [Tri09]. In the quasi-linear variant of the interpretation the cleaning effect of these "weak" flags is subsumed by the use of common definition contexts and thus they become redundant.

The fully uniform quantifier $\overset{\pm}{\forall}$ can be used when the variable x has neither positive nor negative meaning. As explained above, this can be achieved by defining $\sigma^-(\overset{\pm}{\forall} x\, A) := \sigma^-(A)$ and thus discarding x both as a parameter in the shared definition context, and as a constructed counterexample when $\overset{\pm}{\forall} x\, A$ is being used as an assumption.

Example 5.2 explains a case in which we need a negatively uniform quantifier $\overset{-}{\forall}$. However, in order to interpret it, we need to loosen the connection between its two uses: as a type of an extracted counterexample, and as a argument type in the definition context. We would like to be able to disable the former, while keeping the latter. This can be done by introducing a *semi-negative computational type* $\sigma^{\frown}(A)$, which collects the components discarded by the negative uniform flags in $\sigma^-(A)$. The definition context will still use $\sigma^-(A)$, but the "asterisk" type $\sigma^*(A)$ will be extended to depend on an additional parameter of type $\sigma^{\frown}(A)$, which will keep the positive use of the discarded components. On the other hand, the extracted context-dependent counterexample programs $\llbracket M \rrbracket_i^-$ will be of the fully negative computational type $\sigma^-(C_i)$, which takes into account components discarded by any uniform flags.

5.3 Uniform annotations

We consider a system $\overline{\mathrm{NA}^\omega}$, which extends the formula language of NA^ω by allowing different variants of the connectives $\forall$ and $\to$ obtained by annotating them with *uniformity flags*. We consider two uniform variants of the universal quantifier: $\overset{-}{\forall}, \overset{\pm}{\forall}$ and five uniform variants of the implication $\xrightarrow{-\quad}$, $\xrightarrow{\quad -}$, $\xrightarrow{-\ -}$, $\xrightarrow{\pm\quad}$, $\xrightarrow{\pm\ -}$. We will use $\overset{\bullet}{\forall}$ and $\overset{\bullet}{\to}$ to denote a connective with some arbitrary (including empty) annotation.

Definition 5.4 (Pure variant)**.** For every formula A in $\overline{\mathrm{NA}^\omega}$ we define its *pure variant* $A^\bullet \in \mathrm{NA}^\omega$ by deleting all uniform annotations in A. Formally,

$$\begin{aligned}
(\mathrm{at}(t))^\bullet &:= \mathrm{at}(t), \\
(A \overset{\bullet}{\to} B)^\bullet &:= A^\bullet \to B^\bullet, \\
(\overset{\bullet}{\forall} x\, B)^\bullet &:= \forall x\, A^\bullet.
\end{aligned}$$

First, let us define how the computational types are changed in the presence of uniformity flags. The negative type $\sigma^-(A)$ will be affected by all uniformity flags. We will introduce the semi-negative computational type, which will collect dependencies for the positive type $\sigma^+(A)$. The positive computational type will need to be modified accordingly in the case of implication.

Definition 5.5. Let A a formula in $\overline{\mathrm{NA}^\omega}$. We define the negative $\sigma^-(A)$, semi-negative $\sigma^\frown(A)$ and positive computational types of A by simultaneous induction. We also define $\sigma^\smile(A) := \sigma^\frown(A) \Rightarrow \sigma^+(A)$ and extend the definition of $\sigma^*(A) := \sigma^-(A) \Rightarrow \sigma^\smile(A)$.

A	$\sigma^-(A)$	$\sigma^\frown(A)$	$\sigma^+(A)$
$\mathrm{at}(t)$	l	l	l
$\forall x^\rho\, B$	$\rho \times \sigma^-(B)$	$\sigma^\frown(B)$	$\sigma^+(B)$
$\overset{-}{\forall} x^\rho\, B$	$\sigma^-(B)$	$\rho \times \sigma^\frown(B)$	
$\overset{\pm}{\forall} x^\rho\, B$	$\sigma^-(B)$	$\sigma^\frown(B)$	
$B \to C$	$\sigma^*(B) \times \sigma^-(C)$	l	$\sigma^\smile(C) \times \sigma^-(B)$
$B \xrightarrow{-} C$	$\sigma^-(C)$	$\sigma^*(B)$	
$B \xrightarrow{-} C$	$\sigma^*(B)$	$\sigma^-(C)$	
$B \xrightarrow{-\,-} C$	l	$\sigma^*(B) \times \sigma^-(C)$	
$B \xrightarrow{\pm} C$	$\sigma^-(C)$	l	
$B \xrightarrow{\pm\,-} C$	l	$\sigma^-(C)$	

Note that by altering the definition of $\sigma^*(A)$ we departed from our goal in Chapter 4 not to use curried functions. However, this change is necessary, as the definition of computational types in Section 4.5 depended on a full duality between the positive and negative computational types. The use of semi-uniform flags destroys this duality and consequently we need to express two different kinds of dependencies, which is reflected in the definition of $\sigma^*(A)$. Even with this division of dependencies the quasi-linear bound will still hold, as we effectively introduce only one new parameter of type $\sigma^\frown(A)$, regardless of the depth of the formula. It is easy to see that Definition 5.5 is indeed an extension of Definition 4.13, as shown by the following proposition.

Proposition 5.6. *For $A \in \mathrm{NA}^\omega$ we have $\sigma^\frown(A) = \mathsf{I}$ and hence $\sigma^\smile(A) = \sigma^+(A)$.*

Proof. Induction on the definition of A. □

Since $\sigma^*(A)$ now has two parameters, we sometimes will need to partially apply to a function of type $\sigma^*(A)$ to a component of its second parameter $\sigma^\smile(A)$. Hence, we introduce a second kind of partial application, defined as follows:

$$f^{\rho \Rightarrow \sigma \Rightarrow \tau} \circ\circ t := \lambda x^\rho \, (f x \circ t), \text{ where } x \text{ is a fresh variable.}$$

We are now ready to define the Dialectica translation on $\overline{\mathrm{NA}^\omega}$.

Definition 5.7. For $A \in \overline{\mathrm{NA}^\omega}$, terms $r : \sigma^*(A)$ and $s : \sigma^\frown(A)$ we extend the Dialectica translation $\llparenthesis A \rrparenthesis^r_s$ as follows:

A	$\llparenthesis A \rrparenthesis^r_s$
$\mathrm{at}(t)$	$\mathrm{at}(t)$
$\forall x^\rho \, B$	$\llparenthesis B\,[x := s_\llcorner]\,\rrparenthesis^{r \circ s_\llcorner}_{s_\lrcorner}$
$\bar{\forall} x^\rho \, B$	$\forall x \, \llparenthesis B \rrparenthesis^{r \circ\circ x}_s$
$\overset{\pm}{\forall} x^\rho \, B$	$\forall x \, \llparenthesis B \rrparenthesis^r_s$
$B \to C$	$\llparenthesis B \rrparenthesis^{s_\llcorner}_{r s_\lrcorner} \to \llparenthesis C \rrparenthesis^{(r \circ s_\llcorner)_\llcorner}_{s_\lrcorner}$
$B \xrightarrow{-} C$	$\forall x \left(\llparenthesis B \rrparenthesis^{x}_{r s x_\lrcorner} \to \llparenthesis C \rrparenthesis^{(r \circ\circ x)_\llcorner}_{s} \right)$
$B \xrightarrow{\quad -} C$	$\forall x \left(\llparenthesis B \rrparenthesis^{s}_{r s x_\lrcorner} \to \llparenthesis C \rrparenthesis^{r s_\llcorner}_{x} \right)$
$B \xrightarrow{-\ -} C$	$\forall x \left(\llparenthesis B \rrparenthesis^{x_\llcorner}_{r x_\lrcorner} \to \llparenthesis C \rrparenthesis^{(r \circ x_\llcorner)_\llcorner}_{x_\lrcorner} \right)$
$B \xrightarrow{\pm} C$	$\forall x \left(\llparenthesis B \rrparenthesis^{x}_{r s_\lrcorner} \to \llparenthesis C \rrparenthesis^{r_\llcorner}_{s} \right)$
$B \xrightarrow{\pm\ -} C$	$\forall x \left(\llparenthesis B \rrparenthesis^{x_\llcorner}_{r (x_\lrcorner)_\lrcorner} \to \llparenthesis C \rrparenthesis^{r_\llcorner}_{x_\lrcorner} \right)$

Proofs in $\overline{\mathrm{NA}^\omega}$ have the same structure as proofs of NA^ω where the only difference is in the formula language being used. All annotated variants of a given connective are introduced and eliminated via the same rules as their original unannotated counterpart. Note that we require that the connectives in the induction axioms $\mathcal{C}, \mathsf{Ind}_\mathsf{N}, \mathsf{Ind}_{\mathsf{L}(\rho)}$ remain unannotated, however they can be now instantiated with any formula $A \in \overline{\mathrm{NA}^\omega}$.

In order to prove soundness of uniform annotations, we need to impose appropriate restrictions regarding the annotated connectives. There will be two types of restrictions:

1. decidability conditions for Dialectica translations on which a case distinction is needed, and
2. variable conditions for extracted terms on introduction rules.

The first set of conditions can be imposed solely on the syntactic form of assumptions.

Definition 5.8 (Partially uniform formula)**.** A formula $C \in \overline{\mathrm{NA}^\omega}$ is *partially uniform* if $C \neq C^\bullet$ and $\sigma^-(C) \neq \mathsf{I}$. Note that a formula is *not* partially uniform when it is has no uniform annotations ($C \in \mathrm{NA}^\omega$) or when it has enough uniform annotations to not require challenges ($\sigma^-(C) = \mathsf{I}$).

Definition 5.9. A proof $\mathcal{P}$ in $\overline{\mathrm{NA}^\omega}$ is *uniformly interpretable* if for every needed case distinction $\overset{u:C}{\bowtie}$, the formula C has no uniform annotations. Formally, we require that for every subproof M of $\mathcal{P}$:

1. when $M = M_1 M_2$, every shared assumption $u^C \in \mathsf{FA}(M_1) \cap \mathsf{FA}(M_2)$ is not partially uniform;
2. when $M = \mathsf{Ind}_{\mathsf{N}}^{n,A}\, n\, M_1\, (\lambda n\, \lambda u_0\, M_2)$ or $M = \mathsf{Ind}_{\mathsf{L}(\rho)}^{l,A}\, l\, M_1\, (\lambda x\, \lambda l\, \lambda u_0\, M_2)$, every step assumption $u^C \in \mathsf{FA}(M_2)$ is not partially uniform.

The second set of uniformity conditions depends on the notion of extracted terms in proofs in $\overline{\mathrm{NA}^\omega}$. Hence, we will first extend the definition of extracted terms to the annotations by defining the following three components:

1. a definition context $[\![M]\!] : \sigma^-(A) \Rightarrow \diamond$
2. a context-dependent positive witnessing term $[\![M]\!]^+ : \sigma^{\smile}(A)$
3. context-dependent negative witnessing terms $[\![M]\!]_i^- : \sigma^-(C_i)$

In order to ensure correctness of the construction $\overset{u}{\bowtie}$, we need to require that the proof M is uniformly interpretable. The definition of the extracted terms is summarised in Table 5.2. As in Theorem 4.23, we assume that $\mathcal{P}^A$ is a proof term in $\overline{\mathrm{NA}^\omega}$ with assumptions among $\{u_i : C_i\}_{i \geq 1}$ and that we have fresh witnessing variables $X = \{x_i : \sigma^*(C_i)\}$, each one associated uniquely with an assumption variable u_i.

Definition 5.10 (Uniformity restrictions)**.** Let $\mathcal{P}$ be a uniformly interpretable proof in $\overline{\mathrm{NA}^\omega}$. $\mathcal{P}$ is *computationally correct* if every introduction of an annotated connective satisfies the following restrictions

rule	flags	restriction
$\lambda x\, M$	$\bar{\forall}$	$x \notin \bigcup \mathsf{FV}(\{\!\|M\|\!\}_i^-\, y)$
	$\overset{\pm}{\forall}$	$x \notin \mathsf{FV}(\{\!\|M\|\!\})$
$\lambda u_0\, M$	$\overset{-\ \ \ }{\longrightarrow}$	$x_0 \notin \bigcup \mathsf{FV}(\{\!\|M\|\!\}_i^-\, y)$
	$\overset{\ \ \ -}{\longrightarrow}$	$y \notin \bigcup \mathsf{FV}(\{\!\|M\|\!\}_i^-\, y)$
	$\overset{-\ -}{\longrightarrow}$	$x_0, y \notin \bigcup \mathsf{FV}(\{\!\|M\|\!\}_i^-\, y)$
	$\overset{\pm\ \ \ }{\longrightarrow}$	$x_0 \notin \mathsf{FV}(\{\!\|M\|\!\})$
	$\overset{\pm\ -}{\longrightarrow}$	$x_0 \notin \mathsf{FV}(\{\!\|M\|\!\})$ $y \notin \bigcup \mathsf{FV}(\{\!\|M\|\!\}_i^-\, y)$

For all restrictions above we consider the *normal forms* of the extracted terms.

$\mathcal{P}$	flags	$\llbracket\mathcal{P}\rrbracket$	$\llbracket\mathcal{P}\rrbracket^+$	$\llbracket\mathcal{P}\rrbracket^-_i$
$\lambda x^\rho\, M$	$\forall$	$\lambda^\circ x\, \llbracket M\rrbracket$	$\llbracket M\rrbracket^+$	$\llbracket M\rrbracket^-_i$
	$\bar{\forall}$	$\llbracket M\rrbracket$	$\lambda^\circ x\, \llbracket M\rrbracket^+$	$\llbracket M\rrbracket^-_i$
	$\overset{\pm}{\forall}$	$\llbracket M\rrbracket$	$\llbracket M\rrbracket^+$	$\llbracket M\rrbracket^-_i$
Mt	$\forall$	$\llbracket M\rrbracket \circ t$	$\llbracket M\rrbracket^+$	$\llbracket M\rrbracket^-_i$
	$\bar{\forall}$	$\llbracket M\rrbracket$	$\llbracket M\rrbracket^+ \circ t$	$\llbracket M\rrbracket^-_i$
	$\overset{\pm}{\forall}$	$\llbracket M\rrbracket$	$\llbracket M\rrbracket^+$	$\llbracket M\rrbracket^-_i$
$\lambda u_0\, M$	$\to$	$\lambda^\circ x_0\, \llbracket M\rrbracket$	$\left\langle \llbracket M\rrbracket^+, \llbracket M\rrbracket^-_0 \right\rangle$	$\llbracket M\rrbracket^-_i$
	$\xrightarrow{-\;\;}$	$\llbracket M\rrbracket$	$\lambda x_0\, \left\langle \llbracket M\rrbracket^+, \llbracket M\rrbracket^-_0 \right\rangle$	$\llbracket M\rrbracket^-_i$
	$\xrightarrow{\;\;-}$	$\lambda x_0\, \textbf{let } f := \{\!\vert M\vert\!\} \textbf{ in } []$	$\lambda y\, \langle f y_\llcorner, f y \rhd 0\rangle$	$(f\square) \rhd i$
	$\xrightarrow{-\;-}$	$\textbf{let } f := \lambda^\circ x_0\, \{\!\vert M\vert\!\} \textbf{ in } []$	$\lambda y\, \langle f y_\llcorner, f y \rhd 0\rangle$	$(f\square) \rhd i$
	$\xrightarrow{\pm\;\;}$	$\llbracket M\rrbracket$	$\left\langle \llbracket M\rrbracket^+, \llbracket M\rrbracket^-_0 \right\rangle$	$\llbracket M\rrbracket^-_i$
	$\xrightarrow{\pm\;-}$	$\textbf{let } f := \{\!\vert M\vert\!\} \textbf{ in } []$	$\lambda y\, \langle f y_\llcorner, f y \rhd 0\rangle$	$(f\square) \rhd i$
$M_1 M_2$	$\to$	$\textbf{let } f := \{\!\vert M_2\vert\!\} \textbf{ in } \llbracket M_1\rrbracket \circ (f_\llcorner)[\textbf{let } z := \llbracket M_1\rrbracket^+ \textbf{ in } []]$	$z_\llcorner$	$\llbracket M_1\rrbracket^-_i \overset{i}{\bowtie} f(z_\lrcorner) \rhd i$
	$\xrightarrow{-\;\;}$	$\textbf{let } f := \{\!\vert M_2\vert\!\} \textbf{ in } \llbracket M_1\rrbracket[\textbf{let } z := \llbracket M_1\rrbracket^+(f_\llcorner) \textbf{ in } []]$	$z_\llcorner$	$\llbracket M_1\rrbracket^-_i \overset{i}{\bowtie} f(z_\lrcorner) \rhd i$
	$\xrightarrow{\;\;-}$	$\lambda y\, \textbf{let } f := \{\!\vert M_2\vert\!\} \textbf{ in } \llbracket M_1\rrbracket(f_\llcorner)[\textbf{let } z := \llbracket M_1\rrbracket^+ y \textbf{ in } []]$	$z_\llcorner$	$\llbracket M_1\rrbracket^-_i \overset{i}{\bowtie} f(z_\lrcorner) \rhd i$
	$\xrightarrow{-\;-}$	$\lambda y\, \textbf{let } f := \{\!\vert M_2\vert\!\} \textbf{ in } \llbracket M_1\rrbracket[\textbf{let } z := \llbracket M_1\rrbracket^+ \langle f_\llcorner, y\rangle \textbf{ in } []]$	$z_\llcorner$	$\llbracket M_1\rrbracket^-_i \overset{i}{\bowtie} f(z_\lrcorner) \rhd i$
	$\xrightarrow{\pm\;\;}$	$\textbf{let } f := \{\!\vert M_2\vert\!\} \textbf{ in } \llbracket M_1\rrbracket[\textbf{let } z := \llbracket M_1\rrbracket^+ \textbf{ in } []]$	$z_\llcorner$	$\llbracket M_1\rrbracket^-_i \overset{i}{\bowtie} f(z_\lrcorner) \rhd i$
	$\xrightarrow{\pm\;-}$	$\lambda y\, \textbf{let } f := \{\!\vert M_2\vert\!\} \textbf{ in } \llbracket M_1\rrbracket[\textbf{let } z := \llbracket M_1\rrbracket^+ y \textbf{ in } []]$	$z_\llcorner$	$\llbracket M_1\rrbracket^-_i \overset{i}{\bowtie} f(z_\lrcorner) \rhd i$

Table 5.2: Extracted terms in $\overline{\mathrm{NA}}^\omega$

5.4 Soundness of uniform annotations

Below we present the soundness theorem of the quasi-linear Dialectica interpretation for the system $\overline{\mathrm{NA}^\omega}$. The only notable difference from Theorem 4.23 is that we consider computationally correct proofs only. We should note that every proof in NA^ω is computationally correct for trivial reasons. Proposition 5.6 guarantees that the present formulation of soundness is indeed an extension of Theorem 4.23.

Theorem 5.11 (Soundness of uniform annotations)**.** *Let $A \in \overline{\mathrm{NA}^\omega}$ be a formula and let $\mathcal{P}^A$ be a* computationally correct *proof term with assumptions among $\{u_i : C_i\}_{i \geq 1}$. Let us have fresh witnessing variables $X = \{x_i : \sigma^*(C_i)\}$, each one associated uniquely with an assumption variable u_i and let $y_A : \sigma^-(A)$ be a fresh challenging variable associated uniquely with the formula A. Then there is a term $\{\!|\mathcal{P}|\!\}$ and a proof $\overline{\mathcal{P}} : (\!|A|\!)_{p_A}^{\{\!|\mathcal{P}|\!\}^+}$, such that*

1. $\mathsf{FA}(\overline{\mathcal{P}}) \subseteq \left\{v_i : (\!|C_i|\!)_{\{\!|\mathcal{P}|\!\}_i^- y_A}^{x_i}\right\}$,
2. $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) \subseteq \mathsf{FV}(\mathcal{P}) \cup X$,
3. $\lceil \{\!|\mathcal{P}|\!\} \rceil \leq K(\lceil \mathcal{P} \rceil [\![\mathcal{P}]\!]^2)$ *for a fixed constant K, not depending on $\mathcal{P}$,*

Proof. First, we note that the definitions of the Dialectica translation $(\!|A|\!)_s^r$ and the extracted terms $\{\!|\mathcal{P}|\!\}$ for the unannotated connectives coincide with the ones considered in Section 4.5. Hence, it is sufficient to only consider the introduction and elimination rules for the uniform annotated connectives; the rest of the cases are proved exactly as in Theorem 4.23.

Case $\lambda x^\rho\, M^B : \overset{\bullet}{\forall} x^\rho\, B$. By induction hypothesis we have a proof of $\overline{M} : (\!|B|\!)_{y_B}^{\{\!|M|\!\}^+}$ with assumptions $w_i : (\!|C_i|\!)_{\{\!|M|\!\}_i^- y_B}^{x_i}$. We can substitute $\xi := [y_B := y_A]$ and then define $\overline{\mathcal{P}} := \lambda x\, \overline{M}\xi$ with $v_i := w_i\xi$. We will show that the definition of $\overline{\mathcal{P}}$ is correct for each of the uniform annotations.

Subcase $\bar{\forall}$. By definition we have $\{\!|\mathcal{P}|\!\}_i^- \equiv \{\!|M|\!\}_i^-$ and

$$(\!|\bar{\forall} x\, B|\!)_{y_A}^{\{\!|\mathcal{P}|\!\}^+} = \forall x\, (\!|B|\!)_{y_A}^{\{\!|\mathcal{P}|\!\}^+ \circ \circ x} = \forall x\, (\!|B|\!)_{y_A}^{\lambda y_B\, [\![\mathcal{P}]\!] y_B [[\![\mathcal{P}]\!]^+ \circ x]} = \forall x\, (\!|B|\!)_{y_A}^{\{\!|M|\!\}^+}.$$

The uniformity condition guarantees that $x \notin \mathsf{FV}(v_i)$, so the universal introduction in $\overline{\mathcal{P}}$ is correct and the free variable condition for $\{\!|\mathcal{P}|\!\}$ is satisfied. Also, $\lceil \{\!|\mathcal{P}|\!\} \rceil \leq \lceil \{\!|M|\!\} \rceil + 6$.

Subcase $\overset{\pm}{\forall}$. By definition $\{\!|\mathcal{P}|\!\} \equiv \{\!|M|\!\}$, hence

$$(\!|\overset{\pm}{\forall} x\, B|\!)_{y_A}^{\{\!|\mathcal{P}|\!\}^+} = \forall x\, (\!|B|\!)_{p_A}^{\{\!|\mathcal{P}|\!\}^+} = \forall x\, (\!|B|\!)_{p_A}^{\{\!|M|\!\}^+}.$$

The uniformity condition guarantees that $x \notin \mathsf{FV}(\{\!|M|\!\})$, hence $x \notin \mathsf{FV}(v_i)$ and the universal introduction in $\overline{\mathcal{P}}$ is correct. The free variable condition and the size bounds trivially hold.

Case $M^{\overset{\bullet}{\forall} x^\rho B} t^\rho$. Let $C := \overset{\bullet}{\forall} x\, B$. By induction hypothesis we have $\overline{M} : (\!|C|\!)_{y_C}^{\{\!|M|\!\}^+}$ with assumptions $w_i : (\!|C_i|\!)_{\{\!|M|\!\}_i^- y_C}^{x_i}$. We can substitute $\xi := [y_B := y_A]$ and then define $\overline{\mathcal{P}} := \overline{M}\xi t$ with $v_i := w_i\xi$. We will show that the definition of $\overline{\mathcal{P}}$ is correct for each of the uniform annotations.

Subcase $\bar{\forall}$. By definition we have $\{\!|\mathcal{P}|\!\}_i^- \equiv \{\!|M|\!\}_i^-$ and

$$(\!|\bar{\forall} x\, B|\!)_{y_A}^{\{\!|M|\!\}^+} = \forall x\, (\!|B|\!)_{y_A}^{\{\!|M|\!\}^+ \circ\circ x} = \forall x\, (\!|B|\!)_{y_A}^{\lambda y_B\, [\![M]\!] y_B [[\![M]\!]^+ \circ x]}.$$

The free variable condition is satisfied since $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) = \mathsf{FV}(\{\!|M|\!\}) \cup \mathsf{FV}(t)$ and we also have $\lceil \{\!|\mathcal{P}|\!\} \rceil \leq \lceil \{\!|M|\!\} \rceil + \lceil t \rceil + 4$.

Subcase $\overset{\pm}{\forall}$. By definition $\{\!|\mathcal{P}|\!\} \equiv \{\!|M|\!\}$, hence

$$(\!|\overset{\pm}{\forall} x\, B|\!)_{y_A}^{\{\!|M|\!\}^+} = \forall x\, (\!|B|\!)_{p_A}^{\{\!|M|\!\}^+} = \forall x\, (\!|B|\!)_{p_A}^{\{\!|\mathcal{P}|\!\}^+}.$$

The free variable condition and the size bounds trivially hold.

Case $\lambda u_0^B\, M^C : B \overset{\bullet}{\to} C$. By induction hypothesis we have a proof term $\overline{M} : (\!|C|\!)_{y_C}^{\{\!|M|\!\}^+}$ with assumptions $w_0 : (\!|B|\!)_{\{\!|M|\!\}_0^- y_C}^{x_0}$ and $w_i : (\!|C_i|\!)_{\{\!|M|\!\}_i^- y_C}^{x_i}$ for $i \geq 1$.

Subcase $\overset{-}{\longrightarrow}$. By definition we have $\{\!|\mathcal{P}|\!\}_i^- \equiv \{\!|M|\!\}_i^-$ for $i \geq 1$ and

$$\begin{aligned}
(\!|B \overset{-}{\longrightarrow} C|\!)_{y_A}^{\{\!|\mathcal{P}|\!\}^+} &= \forall x_0 \left((\!|B|\!)_{\{\!|\mathcal{P}|\!\}^+ y_A x_0 \lrcorner}^{x_0} \to (\!|C|\!)_{y_A}^{(\{\!|\mathcal{P}|\!\}^+ \circ\circ x_0)\llcorner} \right) \\
&= \forall x_0 \left((\!|B|\!)_{\{\!|\mathcal{P}|\!\}^+ y_A x_0 \lrcorner}^{x} \to (\!|C|\!)_{y_A}^{\lambda y_C\, [\![\mathcal{P}]\!] y_C [[\![\mathcal{P}]\!]^+ x_0]\llcorner} \right) \\
&= \forall x_0 \left((\!|B|\!)_{\{\!|M|\!\}_0^- y_A}^{x} \to (\!|C|\!)_{y_A}^{\{\!|M|\!\}^+} \right).
\end{aligned}$$

We can substitute $\xi := [y_A := y_C]$ and then define $\overline{\mathcal{P}} := \lambda x_0\, (\lambda w_0\, \overline{M})\xi$ with $v_i := w_i\xi$. The uniformity condition guarantees that the universal introduction in $\overline{\mathcal{P}}$ is correct and that $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) \subseteq \mathsf{FV}(\{\!|M|\!\}) \setminus \{x_0\}$. Also, $\lceil \{\!|\mathcal{P}|\!\} \rceil \leq \lceil \{\!|M|\!\} \rceil + 2$.

Subcase $\overset{\bar{}}{\longrightarrow}$. By definition we have

$$\begin{aligned}
\{\!|\mathcal{P}|\!\}_i^- y_A &\overset{r}{=} \textbf{let } x_0 := y_A \textbf{ in let } y_C := \square \textbf{ in } \{\!|M|\!\} y_C \rhd i \\
\text{(by Proposition 4.20)} \qquad &\overset{r}{=} \textbf{let } x_0 := y_A \textbf{ in let } y_C := \square \textbf{ in } \{\!|M|\!\}_i^- y_C \\
\text{(since } y_C \notin \mathsf{FV}(\{\!|M|\!\}_i^- y_C)\text{)} \qquad &\overset{r}{=} \{\!|M|\!\}_i^- y_C \xi, \qquad \text{and} \\
(\!|B \overset{\bar{}}{\longrightarrow} C|\!)_{y_A}^{\{\!|\mathcal{P}|\!\}^+} &= \forall x \left((\!|B|\!)_{\{\!|\mathcal{P}|\!\}^+ y_A x \lrcorner}^{y_A} \to (\!|C|\!)_{x}^{\{\!|\mathcal{P}|\!\}^+ y_A \llcorner} \right) \\
&= \forall x \left((\!|B|\!)_{\{\!|M|\!\}_0^- x \xi}^{y_A} \to (\!|C|\!)_{x}^{\lambda y_C\, \{\!|M|\!\}^+ y_C \xi} \right),
\end{aligned}$$

where $\xi := [x_0 := y_A]$. Hence, we define $\overline{\mathcal{P}} := \lambda y_C\, (\lambda w_0\, \overline{M})\xi$ with $v_i := w_i\xi$. The uniformity condition guarantees that the universal introduction in $\overline{\mathcal{P}}$ is correct. The free variable condition holds since $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) \subseteq \mathsf{FV}(\{\!|M|\!\}) \setminus \{x_0\}$. Also,

$$\begin{gathered}
\lceil \llbracket \mathcal{P} \rrbracket^{+} \rceil \leq \llbracket \mathcal{P} \rrbracket + 10, \qquad \lceil \llbracket \mathcal{P} \rrbracket_i^{-} \rceil \leq \llbracket \mathcal{P} \rrbracket + 4, \\
\lceil \llbracket \mathcal{P} \rrbracket \rceil \leq \lceil \{\!|M|\!\} \rceil + 4, \text{ hence} \\
\lceil \{\!|\mathcal{P}|\!\} \rceil \leq \lceil \{\!|M|\!\} \rceil + \llbracket \mathcal{P} \rrbracket^2 + 5 \llbracket \mathcal{P} \rrbracket + 13,
\end{gathered}$$

assuming that $\lceil \square \rceil = 1$, since we can introduce constants for canonical inhabitants.

Subcase $\xrightarrow{-\,-}$. By definition we have

$$\begin{aligned}
\{\!|\mathcal{P}|\!\}_i^{-} \varepsilon &\overset{r}{=} \textbf{let } x_0 := \square \textbf{ in let } y_C := \square \textbf{ in } \{\!|M|\!\} y_C \triangleright i \\
\text{(by Proposition 4.20)} \qquad &\overset{r}{=} \textbf{let } x_0 := \square \textbf{ in let } y_C := \square \textbf{ in } \{\!|M|\!\}_i^{-} y_C \\
\text{(since } x_0, y_C \notin \mathsf{FV}(\{\!|M|\!\}_i^{-} y_C) \text{)} \qquad &\overset{r}{=} \{\!|M|\!\}_i^{-} y_C, \qquad \text{and} \\
(\!|B \xrightarrow{-\,-} C|\!)_{y_A}^{\{\!|\mathcal{P}|\!\}^{+}} &= \forall x \left((\!|B|\!)_{\{\!|\mathcal{P}|\!\}^{+} x_{\lrcorner}}^{x_{\llcorner}} \to (\!|C|\!)_{x}^{(\{\!|\mathcal{P}|\!\}^{+} \circ x_{\llcorner})_{\llcorner}} \right) \\
&= \forall x \left((\!|B|\!)_{\{\!|M|\!\}_0^{-} y_C \xi}^{x_{\llcorner}} \to (\!|C|\!)_{x_{\lrcorner}}^{\{\!|M|\!\}^{+} \xi} \right),
\end{aligned}$$

where $\xi := [x_0 := x_{\llcorner}]\,[y_C := x_{\lrcorner}]$. Hence, we define $\overline{\mathcal{P}} := \lambda x\,(\lambda w_0\,\overline{M})\xi$ with $v_i := w_i \xi$. The uniformity condition guarantees that the universal introduction in $\overline{\mathcal{P}}$ is correct. The free variable condition holds since $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) \subseteq \mathsf{FV}(\{\!|M|\!\}) \setminus \{x_0\}$. Also,

$$\begin{gathered}
\lceil \llbracket \mathcal{P} \rrbracket^{+} \rceil \leq \llbracket \mathcal{P} \rrbracket + 10, \qquad \lceil \llbracket \mathcal{P} \rrbracket_i^{-} \rceil \leq \llbracket \mathcal{P} \rrbracket + 4, \\
\lceil \llbracket \mathcal{P} \rrbracket \rceil \leq \lceil \{\!|M|\!\} \rceil + 9, \text{ hence} \\
\lceil \{\!|\mathcal{P}|\!\} \rceil \leq \lceil \{\!|M|\!\} \rceil + \llbracket \mathcal{P} \rrbracket^2 + 5 \llbracket \mathcal{P} \rrbracket + 18.
\end{gathered}$$

Subcase $\xrightarrow{\pm}$. By definition we have $\{\!|\mathcal{P}|\!\}^{+}{}_{\llcorner} \overset{r}{=} \{\!|M|\!\}^{+}$, $\{\!|\mathcal{P}|\!\}^{+}{}_{\lrcorner} \overset{r}{=} \{\!|M|\!\}_0^{-}$ and $\{\!|\mathcal{P}|\!\}_i^{-} = \{\!|M|\!\}_i^{-}$ for $i \geq 1$, hence

$$\begin{aligned}
(\!|B \xrightarrow{\pm} C|\!)_{y_A}^{\{\!|\mathcal{P}|\!\}^{+}} &= \forall x \left((\!|B|\!)_{\{\!|\mathcal{P}|\!\}^{+} y_{A\lrcorner}}^{x} \to (\!|C|\!)_{y_A}^{\{\!|\mathcal{P}|\!\}^{+}{}_{\llcorner}} \right) \\
&= \forall x \left((\!|B|\!)_{\{\!|M|\!\}_0^{-} y_A}^{x} \to (\!|C|\!)_{y_A}^{\{\!|M|\!\}^{+}} \right).
\end{aligned}$$

We can substitute $\xi := [y_C := y_A]$ and define $\overline{\mathcal{P}} := \lambda x_0\,(\lambda w_0\,\overline{M})\xi$ with $v_i := w_i \xi$. The uniformity condition guarantees that the universal introduction in $\overline{\mathcal{P}}$ is correct, and the free variable condition and the size bounds trivially hold.

Subcase $\xrightarrow{\pm\,-}$. By definition we have

$$\begin{aligned}
\{\!|\mathcal{P}|\!\}_i^{-} \varepsilon &\overset{r}{=} \textbf{let } y_C := \square \textbf{ in } \{\!|M|\!\} y_C \triangleright i \\
\text{(by Proposition 4.20)} \qquad &\overset{r}{=} \textbf{let } y_C := \square \textbf{ in } \{\!|M|\!\}_i^{-} y_C \\
\text{(since } y_C \notin \mathsf{FV}(\{\!|M|\!\}_i^{-} y_C) \text{)} \qquad &\overset{r}{=} \{\!|M|\!\}_i^{-} y_C, \qquad \text{and} \\
(\!|B \xrightarrow{\pm\,-} C|\!)_{y_A}^{\{\!|\mathcal{P}|\!\}^{+}} &= \forall x \left((\!|B|\!)_{\{\!|\mathcal{P}|\!\}^{+} (x_{\lrcorner})_{\lrcorner}}^{x_{\llcorner}} \to (\!|C|\!)_{x_{\lrcorner}}^{\{\!|\mathcal{P}|\!\}^{+}{}_{\llcorner}} \right) \\
&= \forall x \left((\!|B|\!)_{\{\!|M|\!\}_0^{-} (x_{\lrcorner})}^{x_{\llcorner}} \to (\!|C|\!)_{x_{\lrcorner}}^{\{\!|M|\!\}^{+}} \right),
\end{aligned}$$

We can substitute $\xi := [x_0 := x_\llcorner]\,[y_C := x_\lrcorner]$ and define $\overline{\mathcal{P}} := \lambda x\,(\lambda w_0\,\overline{M})\xi$ with $v_i := w_i\xi$. The uniformity condition guarantees that the universal introduction in $\overline{\mathcal{P}}$ is correct and that the free variable condition holds. Also,

$$\begin{gathered}
\lceil \llbracket \mathcal{P} \rrbracket^+ \rceil \leq \llbracket \mathcal{P} \rrbracket + 10, \qquad \lceil \llbracket \mathcal{P} \rrbracket_i^- \rceil \leq \llbracket \mathcal{P} \rrbracket + 4, \\
\lceil \llbracket \mathcal{P} \rrbracket \rceil \leq \lceil \{\!|M|\!\} \rceil + 3, \text{ hence} \\
\lceil \{\!|\mathcal{P}|\!\} \rceil \leq \lceil \{\!|M|\!\} \rceil + \llbracket \mathcal{P} \rrbracket^2 + 5\llbracket \mathcal{P} \rrbracket + 12.
\end{gathered}$$

Case $M_1^{C \overset{\bullet}{\to} A} M_2^C$. Let us define $B := C \overset{\bullet}{\to} A$. By induction hypothesis we have

$$\begin{aligned}
&\overline{M}_1 : \llparenthesis B \rrparenthesis_{y_B}^{\{\!|M_1|\!\}^+} \text{ with assumptions } w_i' : \llparenthesis C_i \rrparenthesis^{x_i}_{\{\!|M_1|\!\}_i^- y_B} \text{ and} \\
&\overline{M}_2 : \llparenthesis C \rrparenthesis_{y_C}^{\{\!|M_2|\!\}^+} \text{ with assumptions } w_i'' : \llparenthesis C_i \rrparenthesis^{x_i}_{\{\!|M_2|\!\}_i^- y_C}.
\end{aligned}$$

Regardless of the uniform annotation, $\{\!|\mathcal{P}|\!\}_i^- y_A$ will always have the form $E[t_1 \overset{C_i}{\bowtie} t_2]$. Since the proof $\mathcal{P}$ is unformly interpretable, the formula C_i is not partially uniform, which guarantees that we can still soundly apply Lemma 4.22. Following the proof of Theorem 4.23, let us assume that by Lemma 4.22 we have proof terms $\mathcal{Q}_i^{(j)}$: $\llparenthesis C_i \rrparenthesis^{x_i}_{t_1 \overset{i}{\bowtie} t_2} \to \llparenthesis C_i \rrparenthesis^{x_i}_{t_j}$ for $j = 1, 2$. By Proposition 4.5 we have a substitution Ξ, which is such that $E[t] = t\Xi$. Thus we obtain $\mathcal{Q}_i^{(j)}\Xi : \llparenthesis C_i \rrparenthesis^{x_i}_{E[t_1 \overset{i}{\bowtie} t_2]} \to \llparenthesis C_i \rrparenthesis^{x_i}_{E[t_j]}$. Finally, we define $\eta_{j,i} := \left[w_i^{(j)} := \mathcal{Q}_i^{(j)}\Xi v_i\right]$. We will make use of those notations during the proof of each of the subcases. The free variable condition and the size bounds will hold as in Theorem 4.23, so we will focus only on validity.

Subcase $\overset{-}{\longrightarrow}$. By definition and Proposition 4.20 we have

$$\begin{aligned}
\llparenthesis C \overset{-}{\longrightarrow} A \rrparenthesis_{y_B}^{\{\!|M_1|\!\}^+} &= \forall x \left(\llparenthesis C \rrparenthesis^x_{\{\!|M_1|\!\}^+ y_B x_\lrcorner} \to \llparenthesis A \rrparenthesis_{y_B}^{(\{\!|M_1|\!\}^+ \circ\circ x)_\llcorner} \right) \\
&= \forall x \left(\llparenthesis C \rrparenthesis^x_{\{\!|M_1|\!\}^+ y_B x_\lrcorner} \to \llparenthesis A \rrparenthesis_{y_B}^{\lambda y_B\, \llbracket M_1 \rrbracket y_B [\llbracket M_1 \rrbracket^+ x]_\llcorner} \right), \\
\{\!|\mathcal{P}|\!\}^+ &\overset{r}{=} \llbracket M_1 \rrbracket [\llbracket M_1 \rrbracket^+ \{\!|M_2|\!\}^+{}_\llcorner] \overset{r}{=} \{\!|M_1|\!\}^+ \{\!|M_2|\!\}^+{}_\llcorner, \\
\{\!|\mathcal{P}|\!\}_i^- y_A &\overset{r}{=} E[t_1 \overset{C_i}{\bowtie} t_2], \text{ where} \\
&\quad E := \llbracket M_1 \rrbracket y_A, \quad t_1 := \llbracket M_1 \rrbracket_i^-, \quad t_2 := \{\!|M_2|\!\}_i^- (\llbracket M_1 \rrbracket^+ \{\!|M_2|\!\}^+{}_\lrcorner), \\
E[t_1] &\overset{r}{=} \{\!|M_1|\!\}_i^- y_A, \\
E[t_2] &\overset{r}{=} \{\!|M_2|\!\}_i^- (\{\!|M_1|\!\}^+ y_A \{\!|M_2|\!\}^+{}_\lrcorner).
\end{aligned}$$

We will thus use the substitutions

$$\begin{aligned}
\xi_1 &:= [y_B := y_A] && \text{for } M_1, \\
\xi_2 &:= \left[y_C := \{\!|M_1|\!\}^+ y_A \{\!|M_2|\!\}^+{}_\lrcorner\right] && \text{for } M_2,
\end{aligned}$$

and define

$$\overline{\mathcal{P}} := \overline{\mathcal{P}}_1 \{\!|M_2|\!\}^+ \overline{\mathcal{P}}_2, \qquad \text{where } \overline{\mathcal{P}}_j := \overline{M}_j \xi_j \vec{\eta}_j \text{ for } j = 1, 2.$$

Subcase $\xrightarrow{-}$. By definition and Proposition 4.20 we have

$$\begin{aligned}
(\!|C \xrightarrow{-} A|\!)^{\{\!|M_1|\!\}^+}_{y_B} &= \forall x \left((\!|C|\!)^{y_B}_{\{\!|M_1|\!\}^+ y_B x_\lrcorner} \to (\!|A|\!)^{\{\!|M_1|\!\}^+ y_{B\llcorner}}_x \right) \\
\{\!|\mathcal{P}|\!\}^+ &\overset{r}{=} \lambda y_A \, [\![M_1]\!] \{\!|M_2|\!\}^+ [[\![M_1]\!]^+ y_{A\llcorner}] \overset{r}{=} \{\!|M_1|\!\}^+ \{\!|M_2|\!\}^+ {}_\llcorner, \\
\{\!|\mathcal{P}|\!\}^-_i \, y_A &\overset{r}{=} E[t_1 \overset{C_i}{\bowtie} t_2], \text{ where} \\
&\quad E := [\![M_1]\!] \{\!|M_2|\!\}^+, \quad t_1 := [\![M_1]\!]^-_i, \quad t_2 := \{\!|M_2|\!\}^-_i ([\![M_1]\!]^+ y_{A\lrcorner}), \\
E[t_1] &\overset{r}{=} \{\!|M_1|\!\}^-_i \{\!|M_2|\!\}^+, \\
E[t_2] &\overset{r}{=} \{\!|M_2|\!\}^-_i (\{\!|M_1|\!\}^+ \{\!|M_2|\!\}^+ y_{A\lrcorner}).
\end{aligned}$$

We will thus use the substitutions

$$\begin{aligned}
\xi_1 &:= \left[y_B := \{\!|M_2|\!\}^+ \right] && \text{for } M_1, \\
\xi_2 &:= \left[y_C := \{\!|M_1|\!\}^+ \{\!|M_2|\!\}^+ y_{A\lrcorner} \right] && \text{for } M_2,
\end{aligned}$$

and define

$$\overline{\mathcal{P}} := \overline{\mathcal{P}}_1 y_A \overline{\mathcal{P}}_2, \qquad \text{where } \overline{\mathcal{P}}_j := \overline{M}_j \xi_j \vec{\eta}_j \text{ for } j = 1, 2.$$

Subcase $\xrightarrow{-\,-}$. By definition and Proposition 4.20 we have

$$\begin{aligned}
(\!|C \xrightarrow{-\,-} A|\!)^{\{\!|M_1|\!\}^+}_{y_B} &= \forall x \left((\!|C|\!)^{x_\llcorner}_{\{\!|M_1|\!\}^+ x_\lrcorner} \to (\!|A|\!)^{(\{\!|M_1|\!\}^+ \circ x_\llcorner)_\llcorner}_{x_\lrcorner} \right), \\
\{\!|\mathcal{P}|\!\}^+ &\overset{r}{=} \lambda y_A \, [\![M_1]\!] [[\![M_1]\!]^+ \left\langle \{\!|M_2|\!\}^+, y_A \right\rangle {}_\llcorner] \overset{r}{=} (\{\!|M_1|\!\}^+ \circ \{\!|M_2|\!\}^+)_\llcorner, \\
\{\!|\mathcal{P}|\!\}^-_i \, y_A &\overset{r}{=} E[t_1 \overset{C_i}{\bowtie} t_2], \text{ where} \\
&\quad E := [\![M_1]\!], \quad t_1 := [\![M_1]\!]^-_i, \quad t_2 := \{\!|M_2|\!\}^-_i ([\![M_1]\!]^+ \left\langle \{\!|M_2|\!\}^+, y_A \right\rangle {}_\lrcorner), \\
E[t_1] &\overset{r}{=} \{\!|M_1|\!\}^-_i, \\
E[t_2] &\overset{r}{=} \{\!|M_2|\!\}^-_i (\{\!|M_1|\!\}^+ \left\langle \{\!|M_2|\!\}^+, y_A \right\rangle {}_\lrcorner).
\end{aligned}$$

We will thus use the substitutions

$$\begin{aligned}
\xi_1 &:= [] && \text{for } M_1, \\
\xi_2 &:= \left[y_C := \{\!|M_1|\!\}^+ \left\langle \{\!|M_2|\!\}^+, y_A \right\rangle {}_\lrcorner \right] && \text{for } M_2,
\end{aligned}$$

and define

$$\overline{\mathcal{P}} := \overline{\mathcal{P}}_1 \left\langle \{\!|M_2|\!\}^+, y_A \right\rangle \overline{\mathcal{P}}_2, \qquad \text{where } \overline{\mathcal{P}}_j := \overline{M}_j \xi_j \vec{\eta}_j \text{ for } j = 1, 2.$$

Subcase $\xrightarrow{\pm}$. By definition and Proposition 4.20 we have

$$\begin{aligned}
(\!|C \xrightarrow{\pm} A|\!)_{y_B}^{\{\!|M_1|\!\}^+} &= \forall x \left((\!|C|\!)^{x}_{\{\!|M_1|\!\}^+ y_B \lrcorner} \to (\!|A|\!)^{\{\!|M_1|\!\}^+ \llcorner}_{y_B} \right), \\
\{\!|\mathcal{P}|\!\}^+ &\overset{r}{=} \llbracket M_1 \rrbracket [\llbracket M_1 \rrbracket^+ \llcorner] \overset{r}{=} \{\!|M_1|\!\}^+ \llcorner, \\
\{\!|\mathcal{P}|\!\}^-_i y_A &\overset{r}{=} E[t_1 \overset{C_i}{\bowtie} t_2], \text{ where} \\
&\qquad E := \llbracket M_1 \rrbracket y_A, \quad t_1 := \llbracket M_1 \rrbracket^-_i, \quad t_2 := \{\!|M_2|\!\}^-_i (\llbracket M_1 \rrbracket^+ \lrcorner), \\
E[t_1] &\overset{r}{=} \{\!|M_1|\!\}^-_i y_A, \\
E[t_2] &\overset{r}{=} \{\!|M_2|\!\}^-_i (\{\!|M_1|\!\}^+ y_A \lrcorner).
\end{aligned}$$

We will thus use the substitutions

$$\begin{aligned}
\xi_1 &:= [y_B := y_A] && \text{for } M_1, \\
\xi_2 &:= \left[y_C := \{\!|M_1|\!\}^+ y_A \lrcorner \right] && \text{for } M_2,
\end{aligned}$$

and define

$$\overline{\mathcal{P}} := \overline{\mathcal{P}}_1 \{\!|M_2|\!\}^+ \overline{\mathcal{P}}_2, \qquad \text{where } \overline{\mathcal{P}}_j := \overline{M}_j \xi_j \vec{\eta}_j \text{ for } j = 1, 2.$$

Subcase $\xrightarrow{\pm\,-}$. By definition and Proposition 4.20 we have

$$\begin{aligned}
(\!|C \xrightarrow{\pm\,-} A|\!)_{y_B}^{\{\!|M_1|\!\}^+} &= \forall x \left((\!|C|\!)^{x \llcorner}_{\{\!|M_1|\!\}^+ (x \lrcorner) \lrcorner} \to (\!|A|\!)^{\{\!|M_1|\!\}^+ \llcorner}_{x \lrcorner} \right), \\
\{\!|\mathcal{P}|\!\}^+ &\overset{r}{=} \lambda y_A \, \llbracket M_1 \rrbracket [\llbracket M_1 \rrbracket^+ y_A \llcorner] \overset{r}{=} \{\!|M_1|\!\}^+ \llcorner, \\
\{\!|\mathcal{P}|\!\}^-_i y_A &\overset{r}{=} E[t_1 \overset{C_i}{\bowtie} t_2], \text{ where} \\
&\qquad E := \llbracket M_1 \rrbracket, \quad t_1 := \llbracket M_1 \rrbracket^-_i, \quad t_2 := \{\!|M_2|\!\}^-_i (\llbracket M_1 \rrbracket^+ y_A \lrcorner), \\
E[t_1] &\overset{r}{=} \{\!|M_1|\!\}^-_i, \\
E[t_2] &\overset{r}{=} \{\!|M_2|\!\}^-_i (\{\!|M_1|\!\}^+ y_A \lrcorner).
\end{aligned}$$

We will thus use the substitutions

$$\begin{aligned}
\xi_1 &:= [] && \text{for } M_1, \\
\xi_2 &:= \left[y_C := \{\!|M_1|\!\}^+ y_A \lrcorner \right] && \text{for } M_2,
\end{aligned}$$

and define

$$\overline{\mathcal{P}} := \overline{\mathcal{P}}_1 \left\langle \{\!|M_2|\!\}^+, y_A \right\rangle \overline{\mathcal{P}}_2, \qquad \text{where } \overline{\mathcal{P}}_j := \overline{M}_j \xi_j \vec{\eta}_j \text{ for } j = 1, 2. \qquad \square$$

5.5 Properties of uniform annotations

5.5.1 Separating computational content

The uniform flags can be used as switches to control computational content on a very fine level. By appropriate use of the semi-uniform negative annotations we can completely discard positive content of a given formula, while fully preserving its negative content and vice versa.

Definition 5.12 (Content-discarding translations)**.** Let A be a formula in NA^ω. We define the $\overline{\mathrm{NA}^\omega}$ formulas $A^\oplus$ and $A^\ominus$, which fully discard the positive and negative computational meanings of A respectively, while preserving the opposite content.

$$
\begin{array}{lll}
(\mathrm{at}(t))^\oplus := \mathrm{at}(t) & (\forall z\, B)^\oplus := \forall z\, B^\oplus & (B \to C)^\oplus := B^\ominus \to C^\oplus \\
(\mathrm{at}(t))^\ominus := \mathrm{at}(t) & (\forall z\, B)^\ominus := \bar{\forall} z\, B^\ominus & (B \to C)^\ominus := B \xrightarrow{-\,-} C
\end{array}
$$

Intuitively, in $A^\ominus$ we use semi-uniform flags to inductively discard *only* negative content and then apply this annotation for implication premises in $A^\oplus$.

Proposition 5.13. *Let A be a formula in* NA^ω. *Then*

1. $\sigma^+(A^\oplus) = \sigma^-(A^\ominus) = \mathsf{I}$
2. $\sigma^-(A^\oplus) = \sigma^-(A)$
3. $\sigma^*(A^\ominus) = \sigma^*(A)$

Proof. Simultaneous induction on A. Let us consider only the implication case.

$$
\begin{aligned}
\sigma^+((B \to C)^\oplus) &= \sigma^{\smile}(C^\oplus) \times \sigma^-(B^\ominus) = \mathsf{I} \times \mathsf{I} = \mathsf{I}, \\
\sigma^-((B \to C)^\ominus) &= \sigma^-(B \xrightarrow{-\,-} C) = \mathsf{I}, \\
\sigma^-((B \to C)^\oplus) &= \sigma^*(B^\ominus) \times \sigma^-(C^\oplus) = \sigma^*(B) \times \sigma^-(C) = \sigma^-(B \to C), \\
\sigma^*((B \to C)^\ominus) &= \sigma^-(B \xrightarrow{-\,-} C) \Rightarrow \sigma^{\frown}(B \xrightarrow{-\,-} C) \Rightarrow \sigma^+(B \xrightarrow{-\,-} C) \\
&= \mathsf{I} \Rightarrow (\sigma^*(B) \times \sigma^-(C)) \Rightarrow (\sigma^{\smile}(C) \times \sigma^-(B)) \\
&= \sigma^-(B \to C) \Rightarrow \sigma^+(B \to C) = \sigma^*(B \to C).
\end{aligned}
$$

In the last line we used that by Proposition 5.6 $\sigma^{\smile}(C) = \sigma^+(C)$ since $C \in \mathrm{NA}^\omega$. ☐

Remark 5.14. Note that the definitions of $(B \to C)^\oplus$ and $(B \to C)^\ominus$ are not symmetric. An obvious question is "Why not define $(B \to C)^\ominus := B^\oplus \to C^\ominus$?" Then we would obviously still have $\sigma^-(B^\oplus \to C^\ominus) = \mathsf{I}$. However,

$$
\begin{aligned}
\sigma^*(B^\oplus \to C^\ominus) &= \sigma^-(B^\oplus \to C^\ominus) \Rightarrow \sigma^{\frown}(B^\oplus \to C^\ominus) \Rightarrow \sigma^+(B^\oplus \to C^\ominus) \\
&= \mathsf{I} \Rightarrow \mathsf{I} \Rightarrow \sigma^+(C) \times \sigma^-(B) = \sigma^+(B \to C),
\end{aligned}
$$

which is in general not the same as $\sigma^*(B \to C)$. In other words, we would have discarded the positive contribution of the negative content of the implication, thus failing to fully preserve the positive content of $B \to C$.

An important consequence of using uniform flags in the Dialectica interpretation is that the verifying system is not anymore the quantifier-free fragment NA_0^ω, but the full NA^ω. In other words, the Dialectica translations of $\overline{\mathrm{NA}^\omega}$ formulas have a non-trivial Dialectica translation. The reason for this phenomenon is that the effect of removing positive or negative content is achieved by "pushing" the content from the formula to its translation. The Dialectica translation of a formula with no uniform annotations is always quantifier-free and hence void of any computational meaning. However, the translation of a NA^ω formula might itself have nonempty positive or negative computational type. Thus by applying the interpretation a second time we should be able to partially recover the removed content. We will show that discarded negative content can be fully recovered by a second application of the interpretation. However, for positive content we will not be able to syntactically obtain a witness for A from a witness of $(\!|A^\oplus|\!)_y$. We will show a weaker statement that from a computationally correct proof of $A^\oplus$ we can extract a witness for the original formula A.

Theorem 5.15. *Let A be a formula in NA^ω. Then*

1. $(\!|A^\ominus|\!)_\varepsilon^x \leftrightarrow \forall y\, (\!|A|\!)_y^x$
2. $\sigma^-((\!|A^\oplus|\!)_y^\varepsilon) = \sigma^+((\!|A^\ominus|\!)_\varepsilon^x) = \mathsf{l}$,
3. $\sigma^-((\!|A^\ominus|\!)_\varepsilon^x) = \sigma^-(A)$,
4. $(\!|(\!|A^\ominus|\!)_\varepsilon^x|\!)_y^\varepsilon = (\!|A|\!)_y^x$

Proof. Induction on the definition of A.
Case $\mathrm{at}(t)$. Trivial.
Case $\forall z\, B$. By definition

$$\begin{aligned}
(\!|(\forall z\, B)^\ominus|\!)_\varepsilon^x &= (\!|\bar{\forall} z\, B^\ominus|\!)_\varepsilon^x = \forall z\, (\!|B^\ominus|\!)_\varepsilon^{x \circ z} \\
\text{(by induction hypothesis)} \quad &\leftrightarrow \forall z\, \forall y'\, (\!|B|\!)_{y'}^{x \circ z} \leftrightarrow \forall y\, (\!|B\, [z := y_\llcorner]\, |\!)_{y_\lrcorner}^{x \circ y_\llcorner} \\
&= \forall y\, (\!|\forall z\, B|\!)_y^x, \\
\sigma^-((\!|(\forall z\, B)^\oplus|\!)_y^\varepsilon) &= \sigma^-((\!|\forall z\, B^\oplus|\!)_y^\varepsilon) = \sigma^-((\!|B^\oplus\, [z := y_\llcorner]\, |\!)_{y_\lrcorner}^\varepsilon) = \mathsf{l}, \\
\sigma^+((\!|A^\ominus|\!)_\varepsilon^x) &= \sigma^+(\forall z\, (\!|B^\ominus|\!)_\varepsilon^{x \circ z}) = \mathsf{l}, \\
\sigma^-((\!|(\forall z\, B)^\ominus|\!)_\varepsilon^x) &= \sigma^-(\forall z\, (\!|B^\ominus|\!)_\varepsilon^{x \circ z}) = \rho \times \sigma^-((\!|B^\ominus|\!)_\varepsilon^{x \circ z}) \\
\text{(by induction hypothesis)} \quad &= \rho \times \sigma^-(B) = \sigma^-(\forall z\, B), \\
(\!|(\!|(\forall z\, B)^\ominus|\!)_\varepsilon^x|\!)_y^\varepsilon &= (\!|\forall z\, (\!|B^\ominus|\!)_\varepsilon^{x \circ z}|\!)_y^\varepsilon \\
&= (\!|(\!|B^\ominus\, [z := y_\llcorner]\, |\!)_\varepsilon^{x \circ y_\llcorner}|\!)_{y_\lrcorner}^\varepsilon
\end{aligned}$$

$$\text{(by induction hypothesis)} \quad = \llparenthesis B\,[z := y_{\llcorner}]\,\rrparenthesis^{x \circ y_{\llcorner}}_{y_{\lrcorner}} = \llparenthesis \forall z\, B \rrparenthesis^{x}_{y}.$$

Case $B \to C$. By definition

$$\begin{aligned}
\llparenthesis (B \to C)^{\ominus} \rrparenthesis^{x}_{\varepsilon} &= \llparenthesis B \overset{-\,-}{\longrightarrow} C \rrparenthesis^{x}_{\varepsilon} = \forall y \left(\llparenthesis B \rrparenthesis^{y_{\llcorner}}_{x y_{\lrcorner}} \to \llparenthesis C \rrparenthesis^{(x \circ y_{\llcorner})_{\llcorner}}_{y_{\lrcorner}} \right) = \forall y\, \llparenthesis B \to C \rrparenthesis^{x}_{y}, \\
\sigma^{-}(\llparenthesis (B \to C)^{\oplus} \rrparenthesis^{\varepsilon}_{y}) &= \sigma^{-}(\llparenthesis B^{\ominus} \rrparenthesis^{y_{\llcorner}}_{\varepsilon} \to \llparenthesis C^{\oplus} \rrparenthesis^{\varepsilon}_{y_{\lrcorner}}) \\
&= \sigma^{*}(\llparenthesis B^{\ominus} \rrparenthesis^{y_{\llcorner}}_{\varepsilon}) \times \sigma^{-}(\llparenthesis C^{\oplus} \rrparenthesis^{\varepsilon}_{y_{\lrcorner}}) = \mathsf{I} \times \mathsf{I} = \mathsf{I}, \\
\sigma^{+}(\llparenthesis (B \to C)^{\ominus} \rrparenthesis^{x}_{\varepsilon}) &= \sigma^{+}(\forall y\, \llparenthesis B \to C \rrparenthesis^{x}_{y}) = \sigma^{+}(\llparenthesis B \to C \rrparenthesis^{x}_{y}) = \mathsf{I}, \\
\sigma^{-}(\llparenthesis (B \to C)^{\ominus} \rrparenthesis^{x}_{\varepsilon}) &= \sigma^{-}(\forall y\, \llparenthesis B \to C \rrparenthesis^{x}_{y}) = (\sigma^{*}(B) \times \sigma^{-}(C)) \times \mathsf{I} = \sigma^{-}(B \to C), \\
\llparenthesis \llparenthesis (B \to C)^{\ominus} \rrparenthesis^{x}_{\varepsilon} \rrparenthesis^{\varepsilon}_{y} &= \llparenthesis \forall y\, \llparenthesis B \to C \rrparenthesis^{x}_{y} \rrparenthesis^{\varepsilon}_{y} = \llparenthesis \llparenthesis B \to C \rrparenthesis^{x}_{y} \rrparenthesis^{\varepsilon}_{\varepsilon} = \llparenthesis B \to C \rrparenthesis^{x}_{y}.
\end{aligned}$$

Note that none of the claims about $(B \to C)^{\ominus}$ required the induction hypothesis. □

In order to show that we can recover positive content as well, we will need to prove a more general statement allowing open assumptions.

Lemma 5.16. Let $\mathcal{P}$ be a proof of $\llparenthesis A^{\oplus} \rrparenthesis^{\varepsilon}_{y_A}$ with assumptions among $\{u_i : \llparenthesis C_i^{\ominus} \rrparenthesis^{x_i}_{\varepsilon}\}_{i \geq 1}$ for a set of fresh witnessing variables $X = \{x_i : \sigma^{*}(C_i)\}$, each one associated uniquely with an assumption variable u_i and a fresh challenge variable $y_A : \sigma^{-}(A)$ associated uniquely with the formula A. Then there is a term $\{\!|\mathcal{P}^{\oplus}|\!\}$ and a proof $\overline{\mathcal{P}^{\oplus}} : \llparenthesis A \rrparenthesis^{\{\!|\mathcal{P}^{\oplus}|\!\}^{+}}_{y_A}$ in NA^{ω}_{0}, such that

1. $\mathsf{FA}(\overline{\mathcal{P}^{\oplus}}) \subseteq \{v_i : \llparenthesis C_i \rrparenthesis^{x_i}_{\{\!|\mathcal{P}^{\oplus}|\!\}^{-}_{i}\, y_A}\}$,
2. $\mathsf{FV}(\{\!|\mathcal{P}^{\oplus}|\!\}) \subseteq \mathsf{FV}(\mathcal{P}) \cup X \setminus \{y_A\}$,

Proof. Induction on the definition of $A^{\oplus}$.

Case $\mathrm{at}(t)$. We apply the Dialectica interpretation to the proof $\mathcal{P}$ and by Theorem 4.23 obtain the terms $\{\!|\mathcal{P}|\!\}$ and a proof $\overline{\mathcal{P}}$ of $\mathrm{at}(t)$ with assumptions $v_i : \llparenthesis \llparenthesis C_i^{\ominus} \rrparenthesis^{x_i}_{\varepsilon} \rrparenthesis^{\varepsilon}_{\{\!|\mathcal{P}|\!\}^{-}_{i}\, \varepsilon}$, which by Theorem 5.15 are in fact $v_i : \llparenthesis C_i \rrparenthesis^{x_i}_{\{\!|\mathcal{P}|\!\}^{-}_{i}}$. We thus set $\{\!|\mathcal{P}^{\oplus}|\!\} := \{\!|\mathcal{P}|\!\}$ and $\overline{\mathcal{P}^{\oplus}} := \overline{\mathcal{P}}$. The variable condition is trivially satisfied.

Case $\forall x\, B^{\oplus}$. Let us consider the proof $M := \mathcal{P}\,[y_A := \langle x, y_B \rangle]$ of $\llparenthesis \forall x\, B^{\oplus} \rrparenthesis^{\varepsilon}_{\langle x, y_B \rangle} = \llparenthesis B^{\oplus} \rrparenthesis^{\varepsilon}_{y_B}$. We can apply the induction hypothesis to M and obtain a term $\{\!|M^{\oplus}|\!\}$ and a proof $\overline{M^{\oplus}}$ of $\llparenthesis B \rrparenthesis^{\{\!|M^{\oplus}|\!\}^{+}}_{y_B}$. We thus set $[\![\mathcal{P}^{\oplus}]\!] := \lambda^{\circ} x\, [\![M^{\oplus}]\!]$, $[\![\mathcal{P}^{\oplus}]\!]^{+} := [\![M^{\oplus}]\!]^{+}$, $[\![\mathcal{P}^{\oplus}]\!]^{-}_{i} = [\![M^{\oplus}]\!]^{-}_{i}$. The variable condition is satisfied, since by induction hypothesis $y_B \notin \{\!|M^{\oplus}|\!\}$. Since by definition $\llparenthesis \forall x\, B \rrparenthesis^{\{\!|\mathcal{P}^{\oplus}|\!\}^{+}}_{\langle x, y_B \rangle} = \llparenthesis B \rrparenthesis^{\{\!|M^{\oplus}|\!\}^{+}}_{y_B}$, we can set $\overline{\mathcal{P}^{\oplus}} := \overline{M^{\oplus}}\,[x := y_{A\llcorner}]\,[y_B := y_{A\lrcorner}]$.

Case $B^{\ominus} \to C^{\oplus}$. By definition $\llparenthesis B^{\ominus} \to C^{\oplus} \rrparenthesis^{\varepsilon}_{y_A} = \llparenthesis B^{\ominus} \rrparenthesis^{y_{A\llcorner}}_{\varepsilon} \to \llparenthesis C^{\oplus} \rrparenthesis^{\varepsilon}_{y_{A\lrcorner}}$. We apply the induction hypothesis to the proof $M := \mathcal{P}\,[y_A := \langle x_0, y_C \rangle]\, u_0$ with $u_0 : \llparenthesis B^{\ominus} \rrparenthesis^{x_0}_{\varepsilon}$ a fresh assumption variable. We obtain a term $\{\!|M^{\oplus}|\!\}$ and a proof $\overline{M^{\oplus}}$ of $\llparenthesis C \rrparenthesis^{\{\!|M^{\oplus}|\!\}^{+}}_{y_C}$ with assumptions $v_0 : \llparenthesis B \rrparenthesis^{x_0}_{\{\!|M^{\oplus}|\!\}^{-}_{0}\, y_C}$ and $v_i : \llparenthesis C_i \rrparenthesis^{x_i}_{\{\!|M^{\oplus}|\!\}^{-}_{i}\, y_C}$ for $i \geq 1$. We set $[\![\mathcal{P}^{\oplus}]\!] :=$

$\lambda^\circ x_0 \, [\![M^\oplus]\!]$, $[\![\mathcal{P}^\oplus]\!]^+ := \big\langle [\![M^\oplus]\!]^+, [\![M^\oplus]\!]_0^- \big\rangle$, $[\![\mathcal{P}^\oplus]\!]_i^- := [\![M^\oplus]\!]_i^-$ for $i \geq 1$. The variable condition is satisfied, since by induction hypothesis $y_C \notin \{\!|M^\oplus|\!\}$. Finally, we define $\overline{\mathcal{P}^\oplus} := (\lambda v_0 \, \overline{M^\oplus}) \, [x_0 := y_A \llcorner] \, [y_C := y_A \lrcorner]$, which is a proof of $(\!|B \to C|\!)_{y_A}^{\{\!|\mathcal{P}^\oplus|\!\}^+}$. ☐

Lemma 5.16 is sufficient to show that we can recover the positive content discarded in provable formulas $A^\oplus$.

Theorem 5.17. *Let A be a formula in* NA^ω *and let $\mathcal{P}$ be a computationally correct proof of $A^\oplus$. Let $y : \sigma^-(A)$ be a fresh challenging variable. Then there is a term t such that $y \notin \mathsf{FV}(t)$ and $(\!|A|\!)_y^t$ is provable in* NA_0^ω.

Proof. By Theorem 5.11 we obtain a proof $\overline{\mathcal{P}} : (\!|A^\oplus|\!)_y^\varepsilon$. By Lemma 5.16 we can set $t := \{\!|\mathcal{P}^\oplus|\!\}$, since $y \notin \mathsf{FV}(\{\!|\mathcal{P}^\oplus|\!\})$ and we have a proof $\overline{\mathcal{P}^\oplus}$ of $(\!|A|\!)_y^{\{\!|\mathcal{P}^\oplus|\!\}}$ in NA_0^ω. ☐

5.5.2 Modeling modified realisability

The semi-uniform flags allow us to completely switch off the negative computational meaning of formulas while fully preserving their positive content. Next, we will demonstrate that via a different translation involving the semi-uniform annotations we can disable the negative computational content of formulas in such a way that their positive meaning is the same as with modified realisability. By this we will be effectively able to simulate modified realisability within the Dialectica interpretation.

The modified realisability is defined in the larger system HA^ω, hence we will combine semi-uniform flags with the weak translation from Section 1.4.

Definition 5.18. Let A be a formula in HA^ω. We define its *realisability translation* A° as follows:

$$
\begin{aligned}
(\mathrm{at}(t))^\circ &:= \mathrm{at}(t),\\
(B \to C)^\circ &:= B^\circ \xrightarrow{\;-\;} C^\circ,\\
(\forall x\, B)^\circ &:= \bar{\forall} x\, B^\circ,\\
(B \wedge C)^\circ &:= B^\circ \,\tilde{\wedge}\, C^\circ,\\
(\exists x\, B)^\circ &:= \tilde{\bar{\exists}} x\, B^\circ.
\end{aligned}
$$

Proposition 5.19. *Let A be a formula in* HA^ω. *Then $\sigma^-(A^\circ) = \mathsf{I}$ and hence $\sigma^*(A^\circ) = \sigma^\smile(A^\circ)$.*

Proof. Induction on the definition of A.

$$
\begin{aligned}
\sigma^-((\mathrm{at}(t))^\circ) &= \sigma^-(\mathrm{at}(t)) = \mathsf{I},\\
\sigma^-((B \to C)^\circ) &= \sigma^-(B^\circ \xrightarrow{\;-\;} C^\circ) = \sigma^-(C^\circ) = \mathsf{I},
\end{aligned}
$$

$$\begin{aligned}
\sigma^-((\forall x\, B)^\circ) &= \sigma^-(\bar{\forall} x\, B^\circ) = \sigma^-(B^\circ) = \mathsf{I},\\
\sigma^-((B \wedge C)^\circ) &= \sigma^-((B^\circ \to C^\circ \to \mathrm{F}) \to \mathrm{F}) = \sigma^*(B^\circ \to C^\circ \to \mathrm{F}) = \mathsf{I}, \text{ since}\\
\sigma^+(B^\circ \to C^\circ \to \mathrm{F}) &= (\sigma^-(C^\circ) \times \sigma^\smile(\mathrm{F})) \times \sigma^-(B^\circ) = (\mathsf{I} \times \mathsf{I}) \times \mathsf{I} = \mathsf{I},\\
\sigma^-((\exists x\, B)^\circ) &= \sigma^-(\tilde{\exists} x\, B^\circ) = \sigma^*(\forall x\, \neg B^\circ) = \mathsf{I}, \text{ since}\\
\sigma^+(\forall x\, \neg B^\circ) &= \sigma^+(\neg B^\circ) = \sigma^\smile(\mathrm{F}) \times \sigma^-(B^\circ) = \mathsf{I} \times \mathsf{I} = \mathsf{I}. \qquad \square
\end{aligned}$$

We will observe that the types $\sigma^\smile(A^\circ)$ and $\tau^\circ(A)$ are isomorphic. We will define a two-way term translation $(\cdot)^{\updownarrow\circ}$ transforming terms between the two types. Before presenting the definition, let us establish some useful relations for the types $\sigma^\smile(A^\circ)$.

Proposition 5.20. *Let B, C be formulas in* HA^ω. *Then*

1. $\sigma^\smile((B \to C)^\circ) = \sigma^\smile(B^\circ) \Rightarrow \sigma^\smile(C^\circ)$,
2. $\sigma^\smile((\forall x\, B)^\circ) = \rho \times \sigma^\frown(B^\circ) \Rightarrow \sigma^+(B^\circ)$,
3. $\sigma^\smile((B \wedge C)^\circ) = \sigma^\smile(B^\circ) \times \sigma^\smile(C^\circ)$,
4. $\sigma^\smile((\exists x\, B)^\circ) = \rho \times \sigma^\smile(B^\circ)$.

Proof. By unfolding the definitions, we verify that

$$\begin{aligned}
\sigma^\smile((B \to C)^\circ) &= \sigma^\frown(B^\circ \xrightarrow{-} C^\circ) \Rightarrow \sigma^+(B^\circ \xrightarrow{-} C^\circ)\\
&= \sigma^*(B^\circ) \Rightarrow \sigma^\smile(C^\circ) \times \sigma^-(B^\circ)\\
&= \sigma^\smile(B^\circ) \to \sigma^\smile(C^\circ),\\
\sigma^\smile((\forall x\, B)^\circ) &= \sigma^\frown(\bar{\forall} x\, B^\circ) \Rightarrow \sigma^+(\bar{\forall} x\, B^\circ) = \rho \times \sigma^\frown(B^\circ) \Rightarrow \sigma^+(B^\circ),\\
\sigma^\smile((B \wedge C)^\circ) &= \sigma^+((B^\circ \to C^\circ \to \mathrm{F}) \to \mathrm{F}) = \sigma^-(B^\circ \to C^\circ \to \mathrm{F})\\
&= \sigma^\smile(B^\circ) \times \sigma^\smile(C^\circ),\\
\sigma^\smile((\exists x\, B)^\circ) &= \sigma^+(\neg\forall x\, \neg B^\circ) = \sigma^-(\forall x\, \neg B^\circ)\\
&= \rho \times \sigma^-(\neg B^\circ) = \rho \times \sigma^\smile(B^\circ). \qquad \square
\end{aligned}$$

Definition 5.21 (Realisability transformations)**.** Let A be a formula in HA^ω. By simultaneous induction on A we define transformations $(\cdot)^{\updownarrow\circ}$, which transform terms of the respective computational types, as shown on Figure 5.2.

A	$t^{\uparrow\circ}$	$t^{\downarrow\circ}$
$\mathrm{at}(r)$	ε	ε
$B \to C$	$\lambda x\, (t x^{\downarrow\circ})^{\uparrow\circ}$	$\lambda x\, (t x^{\uparrow\circ})^{\downarrow\circ}$
$\forall x\, B$	$\lambda x\, (t \circ x)^{\uparrow\circ}$	$\lambda^\circ x\, (t x)^{\downarrow\circ}$
$B \wedge C$	$\langle (t_\llcorner)^{\uparrow\circ}, (t_\lrcorner)^{\uparrow\circ} \rangle$	$\langle (t_\llcorner)^{\downarrow\circ}, (t_\lrcorner)^{\downarrow\circ} \rangle$
$\exists x\, B$	$\langle t_\llcorner, (t_\lrcorner)^{\uparrow\circ} \rangle$	$\langle t_\llcorner, (t_\lrcorner)^{\downarrow\circ} \rangle$

It is not hard to see that the two transformations are dual.

$$\begin{tikzcd}
\tau^\circ(A) \arrow[d, bend left, "\circ"] \\
\sigma^\smile(A^\circ) \arrow[u, bend left, "\circ"]
\end{tikzcd}$$

Figure 5.2: Realisability transformations

Lemma 5.22. Let A be a formula in HA^ω. Then for any terms $r : \sigma^\smile(C^\circ)$ and $s : \tau^\circ(C)$ we have $(r^{\uparrow\circ})^{\downarrow\circ} \overset{r}{=} r$ and $(s^{\downarrow\circ})^{\uparrow\circ} \overset{r}{=} s$.

Proof. A syntactic exercise by induction on the definition. $\square$

Theorem 5.23. *Let A be a formula in HA^ω and let $t : \tau^\circ(A)$. Then $(\!| A^\circ |\!)_\varepsilon^{t^{\uparrow\circ}} \leftrightarrow t\ \mathbf{r}\ A$.*

Proof. Induction on the definition of the formula A.

Case $\mathrm{at}(r)$. Trivial.

Case $B \to C$. By definition we have

$$\begin{aligned}(\!|(B \to C)^\circ|\!)_\varepsilon^{t^{\uparrow\circ}} &= (\!|B^\circ \overset{-}{\longrightarrow} C^\circ|\!)_\varepsilon^{t^{\uparrow\circ}} = \forall x\,((\!|B^\circ|\!)_\varepsilon^{x} \to (\!|C^\circ|\!)_\varepsilon^{t^{\uparrow\circ}x}) \\ &\leftrightarrow \forall z\,((\!|B^\circ|\!)_\varepsilon^{z^{\uparrow\circ}} \to (\!|C^\circ|\!)_\varepsilon^{(tz)^{\uparrow\circ}}) \leftrightarrow \forall z\,(z\ \mathbf{r}\ B \to tz\ \mathbf{r}\ C) = t\ \mathbf{r}\ (B \to C).\end{aligned}$$

Case $\forall x\, B$. By definition we have

$$\begin{aligned}(\!|(\forall x\, B)^\circ|\!)_\varepsilon^{t^{\uparrow\circ}} &= (\!|\bar{\forall} x\, B^\circ|\!)_\varepsilon^{t^{\uparrow\circ}} = \forall x\,(\!|B^\circ|\!)_\varepsilon^{t^{\uparrow\circ}\circ x} \\ &= \forall x\,(\!|B^\circ|\!)_\varepsilon^{(tx)^{\uparrow\circ}} \leftrightarrow \forall x\,(tx\ \mathbf{r}\ B) = t\ \mathbf{r}\ \forall x\, B.\end{aligned}$$

Case $B \wedge C$. By definition we have

$$\begin{aligned}(\!|(B \wedge C)^\circ|\!)_\varepsilon^{t^{\uparrow\circ}} &= (\!|(B^\circ \to C^\circ \to \mathrm{F}) \to \mathrm{F}|\!)_\varepsilon^{t^{\uparrow\circ}} = \neg(\!|B^\circ \to C^\circ \to \mathrm{F}|\!)_{t^{\uparrow\circ}}^{\varepsilon} \\ &= (\!|B^\circ|\!)_\varepsilon^{t^{\uparrow\circ}{}_\llcorner}\,\tilde{\wedge}(\!|C^\circ|\!)_\varepsilon^{t^{\uparrow\circ}{}_\lrcorner} \leftrightarrow (t_\llcorner\ \mathbf{r}\ B)\,\tilde{\wedge}(t_\lrcorner\ \mathbf{r}\ C) \leftrightarrow t\ \mathbf{r}\ B \wedge C.\end{aligned}$$

Case $\exists x\, B$. By definition we have

$$\begin{aligned}(\!|(\exists x\, B)^\circ|\!)_\varepsilon^{t^{\uparrow\circ}} &= (\!|\tilde{\exists} x\, B^\circ|\!)_\varepsilon^{t^{\uparrow\circ}} = \neg(\!|\forall x\, \neg B^\circ|\!)_{t^{\uparrow\circ}}^{\varepsilon} = \neg(\!|\neg B^\circ\,[x := t^{\uparrow\circ}{}_\llcorner]\,|\!)_{t^{\uparrow\circ}{}_\lrcorner}^{\varepsilon} \\ &= \neg\neg(\!|B^\circ\,[x := t_\llcorner]\,|\!)_\varepsilon^{(t_\lrcorner)^{\uparrow\circ}} \leftrightarrow t_\lrcorner\ \mathbf{r}\ B\,[x := t_\llcorner] = t\ \mathbf{r}\ \exists x\, B.\end{aligned}$$ $\square$

Remark 5.24. The definition of the translation $(\cdot)^\circ$ is possible in the context of the original Dialectica interpretation, using an appropriate variant of the uniform annotations. If we had not insisted on removing redundant computations via the quasi-linear variant of the interpretation, then we would have obtained a more direct result, namely that $\tau^+(A^\circ) = \tau^\circ(A)$ and $|A|^t_\varepsilon = t \mathbf{\ r\ } A$ instead of isomorphism and equivalence, respectively.

Remark 5.25. In the Dialectica interpretation we can still recover the original uniform quantifiers for modified realisability, as defined by Berger. Namely, we can define $(\forall^{\mathsf{U}} x\, B)^\circ := \overset{\pm}{\forall} x\, B^\circ$ and $(\exists^{\mathsf{U}} x\, B)^\circ := \neg \overset{\pm}{\forall} x\, \neg B^\circ$. Moreover, we can also define a uniform implication, as the one proposed by Ratiu and Schwichtenberg in [RS09], by postulating that $(B \xrightarrow{\mathsf{U}} C)^\circ := B^\circ \xrightarrow{\pm} C^\circ$.

5.6 Case studies revisited

In this section we will revisit two of the case studies and will demonstrate how we can use uniform quantifiers to remove redundant parameters and simplify the extracted programs.

5.6.1 Integer root

The integer root example was formalised as follows:

$$\forall f^{\mathsf{NS}}\, \forall g^{\mathsf{NS}}\, \forall m^{\mathsf{N}}\, (\forall n^{\mathsf{N}}\, (f(gn) > n) \to \neg(f0 > m) \to \tilde{\exists} n^{\mathsf{N}}\, (\neg(fn > m)\, \tilde{\wedge}\, f(\mathsf{S}n) > m)),$$
$$M := \lambda f\, \lambda g\, \lambda m\, \lambda u^{\forall n\, (f(gn)>n)} \lambda v^{\neg(f0<m)} \lambda w^{\forall n\, (\neg(fn>m) \to \neg(f(\mathsf{S}n)>m))}$$
$$\mathsf{Ind}_{n,\neg(fn>m)}\, (gm)\, v\, w\, (um)$$

As noted in Section 3.2, the Dialectica interpretation extracts more information than modified realisability for the integer root example. Namely, apart from a witness for $\tilde{\exists} n$, a counterexample for $\forall n$ was also extracted. If we would like to omit the counterexample from the extracted term, we would need to signify that $\forall n$ has no computational meaning. This can be achieved by using the annotated variant $\bar{\forall} n$:

$$\forall f^{\mathsf{NS}}\, \forall g^{\mathsf{NS}}\, \forall m^{\mathsf{N}}\, (\bar{\forall} n^{\mathsf{N}}\, (f(gn) > n) \to \neg(f0 > m) \to \tilde{\exists} n^{\mathsf{N}}\, (\neg(fn > m)\, \tilde{\wedge}\, f(\mathsf{S}n) > m))$$

The proof M of the statement will be computationally correct, as the assumption $u : \bar{\forall} n^{\mathsf{N}}\, f(gn) > n$ participates only in universal elimination, which is subject to no uniformity restrictions. Note that we can equivalently use the full uniform quantifier $\overset{\pm}{\forall} n$, as the quantified formula $f(gn) > n$ has no positive content, hence the positive contribution of n does not matter.

The (very unlikely) alternative is to preserve only the counterexample and discard the witness. This effect can also be achieved, but we need to use the semi-uniform existential quantifier $\bar{\tilde{\exists}} n := \neg \bar{\forall} n\, \neg$. However, here we have to be more careful, because the assumption $w : \bar{\forall} n\, (\neg(fn > m) \to \neg(f(\mathsf{S}n) > m))$ can no longer be directly used as the induction step, as the induction axiom does not involve any uniform annotations! This obstacle can be easily overcome by replacing w with the proof $\lambda n\, wn : \forall n\, (\neg(fn > m) \to \neg(f(\mathsf{S}n) > m))$, which redundantly eliminates n and reintroduces it computationally. The uniform restrictions are satisfied, since w participates only in universal elimination. Note that w has no computational content, so the induction axiom also has no content anymore. In particular, the only parameter, which is used computationally is m. Thus, we can also use uniform quantifiers $\overset{\pm}{\forall} f$ and $\overset{\pm}{\forall} g$ as follows:

$$\overset{\pm}{\forall} f^{\mathsf{NS}}\, \overset{\pm}{\forall} g^{\mathsf{NS}}\, \forall m^{\mathsf{N}}\, (\forall n^{\mathsf{N}}\, (f(gn) > n) \to \neg(f0 > m) \to \overset{-}{\tilde{\exists}} n^{\mathsf{N}}\, (\neg(fn > m) \mathbin{\tilde{\wedge}} f(\mathsf{S}n) > m))$$

Now the extracted content will be trivial: $\lambda m\, m$. Note that in order to soundly introduce an annotation, we needed to change the proof. This could be avoided, if we introduce uniform variants of the induction axioms, which use uniform quantifiers only and can be used with formulas with no computational content.

5.6.2 Infinite Pigeonhole Principle

In Section 3.3, we noted that the obtained term $\{\!|M|\!\}$ for the Infinite Pigeonhole Principle contained redundant parameters, which unnecessarily polluted the extracted program with computations, which would never be executed. We can now use appropriate uniform annotations to discard these computations and reduce the program. We already tracked the source of the redundancies in the formulas $\mathrm{Decr}(l, n)$, $\mathrm{Same}(l, n)$ and $\mathrm{Col}(q, l, n)$. Let us instead define:

$$\begin{aligned}
\mathrm{Decr}(l, n) &:= \bar{\forall} k\, (k < n \to l_{\mathsf{S}k} < l_k),\\
\mathrm{Same}(l, n) &:= \bar{\forall} k\, (k < n \to f l_k = f l_{\mathsf{S}k}),\\
\mathrm{Col}(q, l, n) &:= \bar{\forall} k\, (k < \mathsf{S}n \to f l_k = q).
\end{aligned}$$

As in the previous example, we could have also equivalently used $\overset{\pm}{\forall} k$, since the formula kernels have no computational content. The uniform quantifiers cause the assumptions v_1 and v_2 in the proofs M and M_{S} to require no challenges. Thus the universal introductions in M'_{S} and M''_{S} are computationally correct and these proofs have no longer any computational meaning. A similar effect can be seen in the proof M_2, where the assumption $u : \mathrm{Col}(q, l, n)$ has no computational meaning and thus the universal introduction of k is computationally correct.

The extraction from the Unbounded Pigeonhole Principle after applying the uniform annotations is displayed in Table 5.3. The optimisations from Chapters 4 and 5 finally allow us to display the full program $\{\!|M|\!\}$ on a single page in Figure 5.3.

$\mathcal{P}$	$\llbracket\mathcal{P}\rrbracket$	$\llbracket\mathcal{P}\rrbracket^+$	$\bullet$	$\llbracket\mathcal{P}\rrbracket^-_\bullet$
$M^{(i)}_{\mathsf{S}}$	$\lambda k\,[]$	ε		
$M_3 := \lambda m\,\lambda w_1\,\lambda w_2\,u_{\mathsf{S}n}(m :: x :: l)v_0 M'_{\mathsf{S}} M''_{\mathsf{S}}$	$\lambda m\,\mathbf{let}\ z_1 := m :: x :: l\ \mathbf{in}\ []$	ε	$u_{\mathsf{S}n}$	z_1
$M_4 := \lambda v_0\,\lambda v_1\,\lambda v_2\,w(\mathsf{S}x)M_3$	$\mathbf{let}\ z_3 := \mathsf{S}x\ \mathbf{in}$ $\mathbf{let}\ z_4 := \{\!\!\mid M_3\mid\!\!\}(x_w z_3)\ \mathbf{in}\ []$	ε	$u_{\mathsf{S}n}$ w	z_4 z_3
$M_5 := \lambda l\,\mathsf{Ind}\,l\,\mathsf{efq}(\lambda x\,\lambda l\,\lambda p'\,M_4)$	$\lambda l\,\mathbf{let}\ z_5 := \mathcal{R}_{\mathsf{L(N)}} l\,\square\,(\lambda x\,\lambda l\,\lambda x_{p'}\,\{\!\!\mid M_4\mid\!\!\})\ \mathbf{in}\ []$	ε	$u_{\mathsf{S}n}, w$	$z_5 \triangleright \bullet$
$M_{\mathsf{S}} := \lambda u_{\mathsf{S}n}\,pM_5$	$\lambda x_{\mathsf{S}n}\,\mathbf{let}\ z_6 := \{\!\!\mid M_5\mid\!\!\}x_p\ \mathbf{in}\ []$	$z_6 \triangleright u_{\mathsf{S}n}$	w	$z_6 \triangleright w$
$M_6 := \lambda m\,\lambda w_1\,\lambda w_2\,u_0(m{:})\mathsf{AxT}(\lambda k\,\mathsf{efq})(\lambda k\,w_2)$	$\lambda m\,[]$	ε	u_0	$m{:}$
$M_0 := \lambda u_0\,w 0 M_6$	$\mathbf{let}\ z_7 := 0\ \mathbf{in}$ $\mathbf{let}\ z_8 := \{\!\!\mid M_6\mid\!\!\}(x_w z_7)\ \mathbf{in}\ []$	z_8	w	z_7
$M_1 := \mathsf{Ind}\,n\,M_0\,(\lambda n\,\lambda p\,M_{\mathsf{S}})$	$\mathbf{let}\ z_9 := \mathcal{R}_{\mathsf{N}}\,n\,\{\!\!\mid M_0\mid\!\!\}\big(\lambda n\,\lambda p\,\mathbf{let}\ x_p := p_{\llcorner}\ \mathbf{in}$ $\llbracket M_{\mathsf{S}}\rrbracket[\big\langle \llbracket M_{\mathsf{S}}\rrbracket^+, p_{\lrcorner} \overset{w}{\bowtie} \llbracket M_{\mathsf{S}}\rrbracket^-_w \big\rangle]\big)\ \mathbf{in}\ []$	$z_{9\llcorner}$	w	$z_{9\lrcorner}$
M_2	$\lambda\langle q, k\rangle\,[]$	ε		
$M_7 := \lambda l\,\lambda v_0\,\lambda v_1\,\lambda v_2\,v l v_0 v_1(M_2 q v_2)$	$\lambda l\,[]$	ε	v	l
$M_8 := \lambda q\,\lambda w\,M_1 M_7$	$\lambda\langle q, x_w\rangle\,\mathbf{let}\ z_{12} := \{\!\!\mid M_7\mid\!\!\}\ \mathbf{in}$ $\llbracket M_1\rrbracket[\mathbf{let}\ z_{13} := \llbracket M_1\rrbracket^+\ \mathbf{in}\ []]$	$\llbracket M_1\rrbracket^-_w$	v	$z_{12} z_{13}$
$M\ := \lambda r\,\lambda f\,\lambda z\,\lambda n\,\lambda v\,L r f z M_8$	$\lambda\langle r, f, n\rangle\,\mathbf{let}\ z_{14} := \{\!\!\mid M_8\mid\!\!\}\ \mathbf{in}$ $\mathbf{let}\ z_{15} := \{\!\!\mid L\mid\!\!\}\,\langle r, f, z_{14\llcorner}\rangle\ \mathbf{in}\ []$	$\langle z_{15\lrcorner}, z_{14\lrcorner}(z_{15\llcorner})\rangle$		

Table 5.3: Quasi-linear extraction from Unbounded Pigeonhole Principle with uniform annotations

$$
\lambda\langle r,f,n\rangle \left(\begin{array}{l}
\textbf{let } z_{14} := \lambda\langle q,x_w\rangle \left(\begin{array}{l}
\textbf{let } z_8 := \lambda n\,\lambda p \left(\begin{array}{l}
\textbf{let } l := p_\llcorner \textbf{ in}\\
\textbf{let } z_5 := \left(\begin{array}{l} \mathcal{R}\, l\ \square\ (\lambda x\,\lambda l\,\lambda x_{p'}\\ \textbf{let } z_3 := \mathsf{S}x \textbf{ in}\\ \langle x_w z_3 :: x :: l, z_3\rangle)\end{array}\right) \textbf{ in}\\
\left\langle z_{5\llcorner}, p_\lrcorner \overset{w}{\bowtie} z_{5\lrcorner}\right\rangle
\end{array}\right) \textbf{ in}\\
\textbf{let } z_9 := \mathcal{R}\, n\ \langle x_w 0{:}, 0\rangle\, z_8 \textbf{ in } \langle z_{9\lrcorner}, z_{9\llcorner}\rangle
\end{array}\right) \textbf{ in}\\
\textbf{let } z_{15} := \left(\begin{array}{l}
\mathcal{R}\, r\, (\lambda\langle f, x_{u_2}\rangle\,\square)\Big(\lambda r\,\lambda x_p\,\lambda\langle f,x_{u_2}\rangle\\
\textbf{let } g_3 := \lambda n_1 \left(\begin{array}{l}
\textbf{let } g_1 := \lambda n_2\ \langle n_1\sqcup n_2, n_1\sqcup n_2\rangle \textbf{ in}\\
\textbf{let } g_2 := \lambda\langle q,x_w\rangle \left(\begin{array}{l}\textbf{let } z_2 := \langle q, \lambda n_2\, n_1 \sqcup x_w n_2\rangle \textbf{ in}\\ \langle x_{u_2} z_2, z_2\rangle\end{array}\right) \textbf{ in}\\
\textbf{let } z_4 := x_p\,\langle f\lceil n_1, g_{2\llcorner}\rangle \textbf{ in } \langle g_1(z_{4\lrcorner})_\llcorner, g_1(z_{4\lrcorner})_\lrcorner, g_2(z_{4\llcorner})_\lrcorner\rangle
\end{array}\right) \textbf{ in}\\
\textbf{let } z_5 := \langle r, g_{3\llcorner}\rangle \textbf{ in}\\
\textbf{let } z_7 := g_3(x_{u_2} z_5) \textbf{ in } \left\langle z_5 \overset{u_2}{\bowtie} z_{7\lrcorner}, z_{7\llcorner}\right\rangle\Big)\ \langle f, z_{14\llcorner}\rangle
\end{array}\right) \textbf{ in}\\
\langle z_{15\lrcorner}, z_{14\lrcorner}(z_{15\llcorner})\rangle
\end{array}\right)
$$

where

$$n_1 \overset{w}{\bowtie} n_2 \overset{a}{=} \textbf{let } m := x_w n_1 \textbf{ in } \mathsf{Cases}\,(T_{\to}(n_1 \le m)(fm = q))\, n_2\, n_1,$$

$$\langle q_1, h_1\rangle \overset{u_2}{\bowtie} \langle q_2, h_2\rangle \overset{a}{=} \textbf{let } n := x_{u_2}\,\langle q_1, h_1\rangle \textbf{ in let } m := h_1 n \textbf{ in } \mathsf{Cases}\,(T_{\to}(n \le m)(fm \ne q_1))\ \langle q_2, h_2\rangle\ \langle q_1, h_1\rangle\,.$$

Figure 5.3: Simplified program extracted from the Unbounded Pigeonhole Principle

CHAPTER

SIX

DIALECTICA INTERPRETATION WITH MARKED COUNTEREXAMPLES

A specific feature of the Dialectica interpretation which allows to embed classical logic into a quantifier-free constructive system is the extraction of counterexamples. In NA^ω proving $\tilde{\exists} x\, A$ amounts to using the assumption $\forall x\, \neg A$ to derive a contradiction. The non-trivial utilisation of classical logic occurs where this assumption is used more than once. In the extracted term this corresponds to a decision between several counterexamples y of the formula A by checking the validity of its quantifier-free translation $|A|^x_y$. An extreme example of this phenomenon is the interpretation of induction, which corresponds to using the induction hypothesis an unbounded number of times. This is reflected by a case distinction on every recursive step in the recursively defined programs for computing counterexamples for open assumptions. However, there is a special case of the induction scheme in which a case analysis on every step is redundant and, moreover, can lead to an unnecessary increase of complexity.

In this chapter we identify the instances of induction in which redundant computations occur and propose a general solution to mitigate the inefficiency by introducing flags, which determine counterexample validity. We prove that the approach is sound and demonstrate its effectiveness on the Infinite Pigeonhole Principle case study. The results in this chapter have been published in [Tri10a].

6.1 A special case of recursion

Let $\mathcal{P} := \mathsf{Ind}_{\mathsf{N}}^{n,A}\, n\, M_1^{A[n:=0]}\, (\lambda n\, \lambda u_0^A\, M_2^{A[n:=\mathsf{S}n]})$ be a proof by induction from assumptions $u_i : C_i$. Consider the case where A requires no challenges, i.e., $\tau^-(A) = \mathsf{I}$. For the sake of simplicity, let us assume that we have only one open assumption $u : C$

and let us omit indices where possible. The soundness Theorem 2.21 for the original Dialectica interpretation leads to the following two programs:

$$\begin{aligned}
&[\![\mathcal{P}]\!]^+ \equiv \mathcal{R}_{\mathsf{N}}\, n\, [\![M_1]\!]^+ \,(\lambda n\, \lambda x_0\, [\![M_2]\!]^+)\\
&[\![\mathcal{P}]\!]^- \equiv \mathcal{R}_{\mathsf{N}}\, n\, [\![M_1]\!]^- \left(\lambda n\, \lambda p\ ([\![M_2]\!]^- \xi) \overset{C}{\bowtie} p\right), \qquad \text{for } \xi := \left[x_0 := [\![\mathcal{P}]\!]^+\right].
\end{aligned}$$

Clearly, the computation of $[\![\mathcal{P}]\!]^-$ is not optimal, because for every occurrence of x_0 in $[\![M_2]\!]^-$, the recursive process for $[\![\mathcal{P}]\!]^+$ is invoked. To avoid this redundancy, we can apply Theorem 4.23 to obtain the following program:

$$\{\!|\mathcal{P}|\!\} \equiv \mathcal{R}_{\mathsf{N}}\, n\, \{\!|M_1|\!\}\big(\lambda n\, \lambda p\, \mathbf{let}\ x_0 := p_{\llcorner}\ \mathbf{in}\ [\![M_2]\!][\big\langle [\![M_2]\!]^+, [\![M_2]\!]^- \overset{C}{\bowtie} p_{\lrcorner}\big\rangle]\big).$$

Combining positive and negative content in a single computation is already an improvement, because we need only one linear recursive process, as opposed to two nested recursions in the program $[\![\mathcal{P}]\!]^-$ above. As a result, a program of lower worst time complexity is obtained.

However, in this special case we can optimise even further. For a fixed n, $\{\!|\mathcal{P}|\!\}^-$ can be seen as performing a linear search for a counterexample for C among the n candidates in the list $L^n := (\{\!|M_1|\!\}^-, (\{\!|M_2|\!\}^- \xi' [n := k])_{k<n-1})$, where $\xi' := \left[x_0 := \{\!|\mathcal{P}|\!\}^+\right]$. Formally,

$$(\!|C|\!)^x_{\{\!|\mathcal{P}|\!\}^-} \leftrightarrow \bigwedge_{k<n} (\!|C|\!)^x_{L^n_k} \quad \text{and} \quad \exists K < n\ \left(\{\!|\mathcal{P}|\!\}^- = L^n_K\right).$$

This situation already occurred in the Integer Root example (Section 3.2) and in the Infinite Pigeonhole Principle (Section 3.3), where the recursive computation of counterexamples corresponded to a linear search. However, as noted before, there is an important factor determining which of the counterexamples will be chosen. The definition of $\bowtie$ is asymmetric in the sense that it performs the case distinction on the Dialectica translation for one of its operands only:

$$t_1 \overset{C,x}{\bowtie} t_2 := \mathbf{let}\ y := t_1\ \mathbf{in}\ \mathsf{Cases}\,(T_C x y) t_2 y.$$

In particular, if t_1 is returned as a result of the case distinction, then we already have the implicit knowledge that t_1 is indeed a valid counterexample. However, if t_2 is returned, we only know that t_1 is *not* a valid counterexample, but we have no information whatsoever about the validity of t_2.

In its current form $\{\!|\mathcal{P}|\!\}^-$ will always return the *last valid counterexample* in the list L^n, i.e., such a K that $\forall k > K\ (\!|C|\!)^x_{L^n_k}$. As already remarked by Troelstra in the foreword of Gödel's original paper [Göd58], a priori there is no particular reason why we should prefer one counterexample to another. We can exploit this fact so that we choose such a counterexample, which can be computed most efficiently.

An ad-hoc idea would be to prefer the *first valid counterexample* from L^n, i.e., to find K such that $\forall k < K\, (\!|C|\!)^x_{L^n_k}$. We can easily achieve this by simply reversing the operands of $\bowtie$ in the definition of $\{\!|\mathcal{P}|\!\}^-$. Unfortunately, this trivial change will not improve the efficiency of the extracted program, because the recursion will still perform n steps, computing all elements from the list L^n and performing $n-1$ case distinctions. In order to remove redundant computation, it is clearly sufficient to terminate the recursion as soon as we find the first index K for which $\neg(\!|C|\!)^x_{L^n_K}$. Although this will not change the *worst time complexity* of the program, it might improve its *average time complexity* when the expected value of K is lower than $O(n)$.

Such an earlier terminating search could be implemented by adding a boolean flag b, which specifies whether a counterexample is already found. This can be done in the following fashion:

$$\{\!|\mathcal{P}|\!\} := \mathcal{R}_{\mathsf{N}}\, n\, \langle \{\!|M_1|\!\}, \mathsf{ff}\rangle \bigl(\lambda n\, \lambda p\, \lambda b\, \textbf{let}\ x_0 := p_{\llcorner}\ \textbf{in}\ [\![M_2]\!][\Bigl\langle [\![M_2]\!]^+, [\![M_2]\!]^- \overset{b,C}{\ltimes} p_{\lrcorner}\Bigr\rangle]\bigr),$$

$$\text{where } t_1 \overset{b,C}{\ltimes} t_2 := \mathsf{Cases}\, b\, \langle t_1, \mathsf{tt}\rangle \bigl(\mathsf{Cases}(T_C x t_1)\langle t_2, \mathsf{ff}\rangle\langle t_1, \mathsf{tt}\rangle\bigr).$$

Note that the assumption $\sigma^-(A) = \mathsf{I}$ is important, otherwise p would be a function, applied to the negative content $[\![M_2]\!]^-_0$ on each recursive step. In the general case a choice would be made among two new counterexample candidates: $[\![M_2]\!]^-_u$ and $p[\![M_2]\!]^-_0{}_{\lrcorner}$, both depending on the variable n. Therefore, the information that a counterexample is found on an earlier step could not be used for early termination of the recursion. On the other hand, in the special case described above we choose among one new candidate $[\![M_2]\!]^-$, which depends on the current value of n, and the previous counterexample $p_{\lrcorner}$.

In the following sections this trick will be generalised so that it can be soundly integrated into the quasi-linear Dialectica interpretation from Chapter 4.

6.2 Counterexample marking

As discussed above, the programs extracted via the original Dialectica interpretation do not take advantage of the information about the validity of a counterexample. The case distinction construction $t_1 \overset{u}{\bowtie} t_2$ forces a choice between two candidate counterexamples t_1 and t_2 for the assumption $u : C$. This choice is made by a direct check of the decidable Dialectica translation of the formula C instantiated with one of the terms t_i. What is not taken into account is that if the check confirms that the chosen candidate is indeed a counterexample, all further computation of witnesses and counterexamples for C is pointless. In a certain sense, this can be viewed as avoiding both

1. *recomputation* — the validity check of the counterexample is repeated if we have more than two occurrences of the assumption C,
2. *redundant computation* — all further counterexamples and witnesses computed are not needed for a sound verification proof.

It is important to note that the definition context approach from Chapter 4 seems inapplicable for avoiding such kind of recomputation. The reason is the underlying difference between repeated subterms and the recomputation considered here. Term duplication can be detected during the *extraction process* and using a shared context is one possible method to avoid it. However, the counterexample decision occurs during the *evaluation of the program* and, depending on the input parameters, recomputation might or might not occur. Attempting to use a shared context would imply precomputation of all possible case distinctions, which could be much worse than recomputing only one case distinction.

We will thus follow a different idea. As was already hinted in Section 6.1, an additional marker will be attached to each extracted counterexample, carrying information about its validity. The type of booleans B will be used as a type for markers. New variants $\rho^+(A)$ and $\rho^-(A)$ of the computational types will be defined to accommodate the marker type by introducing a new *marked computational type*, defined as $\rho^{-\circ}(A) := \mathsf{B} \times \rho^-(A)$. The corresponding reformulation of the Dialectica translation will be denoted as $(\!\|C\|\!)^x_y$.

For clarity, $t \blacktriangleright m := \langle m, t\rangle$ will denote that $t : \rho^-(A)$ is marked by m. Consequently, when we write $t \blacktriangleright m \overset{r}{=} s$, we will mean that $m \overset{r}{=} s_{\llcorner}$ and $t \overset{r}{=} s_{\lrcorner}$. The markers have the following intended meaning:

- $t \blacktriangleright \mathsf{tt}$ — we have no information yet about the validity of $(\!\|C_i\|\!)^{x_i}_t$,
- $t \blacktriangleright \mathsf{ff}$ — we have checked that $\neg(\!\|C_i\|\!)^{x_i}_t$,

Remark 6.1. In this presentation we reduce the three markers suggested in [Tri10a] to two. We have removed the marker signifying that t is an arbitrarily chosen counterexample and its validity need not be checked. The reason is that this marker complicates the interpretation, while its optimisational effect is negligible.

We are ready to incorporate the marker type in the Dialectica negative computational types as follows.

Definition 6.2 (Marked computational types)**.** For a formula A in NA^ω we redefine the positive and negative computational types denoting the new variants as $\rho^+(A)$

and $\rho^-(A)$. We will also denote $\rho^*(A) := \rho^-(A) \Rightarrow \rho^+(A)$ and $\rho^{-\circ}(A) := \mathsf{B} \times \rho^-(A)$. We define:

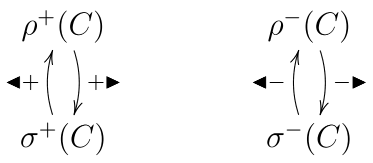
$$\begin{aligned}
&\rho^+(\mathrm{at}(b)) := \varepsilon, && \rho^-(\mathrm{at}(b)) := \varepsilon,\\
&\rho^+(B \to C) := \rho^+(C) \times \rho^{-\circ}(B), && \rho^-(B \to C) := \rho^*(B) \times \rho^-(C)\\
&\rho^+(\forall x^\sigma\, B) := \rho^+(B), && \rho^-(\forall x^\sigma\, B) := \sigma \times \rho^-(B).
\end{aligned}$$

The change in the positive type in the implication case of the translation leads to a slight adjustment to the Dialectica translation:

Definition 6.3 (Dialectica translation with markers)**.** Let A be a formula in NA^ω and let $r : \rho^*(A)$ and $s : \rho^-(A)$ be terms. We define the Dialectica translation of A with counterexample marking as follows, where the difference from the definition of the quasi-linear translation is emphasized by a box below:

$$\begin{aligned}
(\!\|\mathrm{at}(t)\|\!)^\varepsilon_\varepsilon &:= \mathrm{at}(t),\\
(\!\|B \to C\|\!)^r_s &:= (\!\|B\|\!)^{s_\llcorner}_{rs_\lrcorner \boxed{\lrcorner}} \to (\!\|C\|\!)^{(r\circ s_\llcorner)_\llcorner}_{s_\lrcorner},\\
(\!\|\forall x\, B\|\!)^r_s &:= (\!\|B\,[x := s_\llcorner]\,\|\!)^{r\circ s_\llcorner}_{s_\lrcorner}.
\end{aligned}$$

The definition above declares the marker irrelevant for the logical validity of the Dialectica translation of a given formula. Therefore, if we define marker-erasing mappings as shown on Figure 6.1, we will obtain exactly the interpretation $(\!|A|\!)^r_s$.



$$\begin{tikzcd}
\rho^+(C) \arrow[d, bend left, "+\blacktriangleright"] & \rho^-(C) \arrow[d, bend left, "-\blacktriangleright"] \\
\sigma^+(C) \arrow[u, bend left, "\blacktriangleleft+"] & \sigma^-(C) \arrow[u, bend left, "\blacktriangleleft-"]
\end{tikzcd}$$

Figure 6.1: Transformations between marked and unmarked Dialectica types

Definition 6.4 (Marker-erasing transformations)**.** Let A be a formula in NA^ω. By induction on the formula A, we define marker-erasing transformations $(\cdot)^{\pm\blacktriangleright}$ and $(\cdot)^{\blacktriangleleft\pm}$ transforming terms of type $\rho^\pm(A)$ to terms of type $\sigma^\pm(A)$ and vice versa as shown in Figure 6.1.

A	$t^{+\blacktriangleright}$	$t^{-\blacktriangleright}$
$\mathrm{at}(r)$	ε	ε
$B \to C$	$\left\langle (t_\llcorner)^{+\blacktriangleright}, (t_{\lrcorner\lrcorner})^{-\blacktriangleright} \right\rangle$	$\left\langle \lambda y\, (t_\llcorner y^{\blacktriangleleft-})^{+\blacktriangleright}, (t_\lrcorner)^{-\blacktriangleright} \right\rangle$
$\forall x\, B$	$t^{+\blacktriangleright}$	$\left\langle t_\llcorner, (t_\lrcorner)^{-\blacktriangleright} \right\rangle$

A	$t^{\blacktriangleleft +}$	$t^{\blacktriangleleft -}$
$\mathrm{at}(r)$	ε	ε
$B \to C$	$\left\langle (t_{\llcorner})^{\blacktriangleleft +}, (t_{\lrcorner})^{\blacktriangleleft -} \blacktriangleright \mathsf{tt} \right\rangle$	$\left\langle \lambda y\, (t_{\llcorner} y^{- \blacktriangleright})^{\blacktriangleleft +}, (t_{\lrcorner})^{\blacktriangleleft -} \right\rangle$
$\forall x\, B$	$t^{\blacktriangleleft +}$	$\left\langle t_{\llcorner}, (t_{\lrcorner})^{\blacktriangleleft -} \right\rangle$

Lemma 6.5. Let A be a formula in NA^{ω}. Then for any terms $r : \sigma^{+}(A)$ and $s : \sigma^{-}(A)$ we have $(r^{+\blacktriangleright})^{\blacktriangleleft +} \overset{r}{=} r$ and $(s^{-\blacktriangleright})^{\blacktriangleleft -} \overset{r}{=} s$.

Proof. A syntactic exercise by induction on the definition. Note that for $r : \rho^{+}(A)$ and $s : \rho^{-}(A)$ the dual equalities $(r^{\blacktriangleleft +})^{+\blacktriangleright} \overset{r}{=} r$ and $(s^{\blacktriangleleft -})^{-\blacktriangleright} \overset{r}{=} s$ do not hold since a deleted marker cannot be restored. ☐

Proposition 6.6. *Let A be a formula in* NA^{ω}. *Then for any terms $r : \rho^{*}(A)$ and $s : \rho^{-}(A)$ we have* $\llparenthesis\!| A |\!\rrparenthesis^{r}_{s} = \llparenthesis A \rrparenthesis^{\lambda y\, (r y^{\blacktriangleleft -})^{+\blacktriangleright}}_{s^{-\blacktriangleright}}$.

Proof. Induction on A.

Case $\mathrm{at}(t)$. Trivial.

Case $B \to C$. By definition and Lemma 6.5:

$$\begin{aligned}
\llparenthesis B \to C \rrparenthesis^{\lambda y\, (r y^{\blacktriangleleft -})^{+\blacktriangleright}}_{s^{-\blacktriangleright}} &= \llparenthesis B \rrparenthesis^{s^{-\blacktriangleright}{}_{\llcorner}}_{(r(s^{-\blacktriangleright})^{\blacktriangleleft -})^{+\blacktriangleright}{}_{\lrcorner}} \to \llparenthesis C \rrparenthesis^{\lambda z\, (r(\langle s^{-\blacktriangleright}{}_{\llcorner}, z\rangle)^{\blacktriangleleft -})^{+\blacktriangleright}{}_{\llcorner}}_{s^{-\blacktriangleright}{}_{\lrcorner}} \\
&= \llparenthesis B \rrparenthesis^{\lambda y\, (s_{\llcorner} y^{\blacktriangleleft -})^{+\blacktriangleright}}_{(rs)^{+\blacktriangleright}{}_{\lrcorner}} \to \llparenthesis C \rrparenthesis^{\lambda z\, (r\langle s_{\llcorner}, z^{\blacktriangleleft -}\rangle_{\llcorner})^{+\blacktriangleright}}_{(s_{\lrcorner})^{-\blacktriangleright}} \\
&= \llparenthesis B \rrparenthesis^{\lambda y\, (s_{\llcorner} y^{\blacktriangleleft -})^{+\blacktriangleright}}_{(r s_{\lrcorner\lrcorner})^{-\blacktriangleright}} \to \llparenthesis C \rrparenthesis^{\lambda z\, ((r \circ s_{\llcorner})_{\llcorner} z^{\blacktriangleleft -})^{+\blacktriangleright}}_{(s_{\lrcorner})^{-\blacktriangleright}} \\
\text{(by induction hypothesis)} \quad &= \llparenthesis\!| B |\!\rrparenthesis^{s_{\llcorner}}_{r s_{\lrcorner\lrcorner}} \to \llparenthesis\!| C |\!\rrparenthesis^{(r \circ s_{\llcorner})_{\llcorner}}_{s_{\lrcorner}} = \llparenthesis\!| B \to C |\!\rrparenthesis^{r}_{s}.
\end{aligned}$$

Case $\forall x\, B$. By definition and Lemma 6.5:

$$\begin{aligned}
\llparenthesis \forall x\, B \rrparenthesis^{\lambda y\, (r y^{\blacktriangleleft -})^{+\blacktriangleright}}_{s^{-\blacktriangleright}} &= \llparenthesis B\left[x := s^{-\blacktriangleright}{}_{\llcorner}\right] \rrparenthesis^{\lambda z\, (r(\langle s^{-\blacktriangleright}{}_{\llcorner}, z\rangle)^{\blacktriangleleft -})^{+\blacktriangleright}}_{s^{-\blacktriangleright}{}_{\lrcorner}} \\
&= \llparenthesis B\left[x := s_{\llcorner}\right] \rrparenthesis^{\lambda z\, (r\langle s_{\llcorner}, z^{\blacktriangleleft -}\rangle)^{+\blacktriangleright}}_{(s_{\lrcorner})^{-\blacktriangleright}} \\
&= \llparenthesis B\left[x := s_{\llcorner}\right] \rrparenthesis^{\lambda z\, ((r \circ s_{\llcorner}) z^{\blacktriangleleft -})^{+\blacktriangleright}}_{(s_{\lrcorner})^{-\blacktriangleright}} \\
\text{(by induction hypothesis)} \quad &= \llparenthesis\!| B\left[x := s_{\llcorner}\right] |\!\rrparenthesis^{r \circ s_{\llcorner}}_{s_{\lrcorner}} = \llparenthesis\!| \forall x\, B |\!\rrparenthesis^{r}_{s}.
\end{aligned}$$

☐

6.3 Soundness of counterexample marking

As could be expected, the essential use of counterexample markers comes in the definition of case distinction terms. In order to use the information carried by the marker, we need to have additional assumptions, which reflect the semantics of the marker as given in Section 6.2. Thus a marker ff has to imply a false Dialectica translation and the marker $\mathtt{tt}$ carries a neutral meaning, thus its presence has to equate the case distinction terms to those defined in Lemma 4.22.

Lemma 6.7. There is a constant K, such that for every formula C there is a term $T_C : \rho^*(C) \Rightarrow \rho^-(C) \Rightarrow \mathsf{B}$ such that:

1. $(\!|C|\!)^r_s \leftrightarrow \mathrm{at}(T_C rs)$
2. $\lceil T_C \rceil \leq K \lceil C \rceil$

Proof. Similar to Lemma 4.21, but adjusted for the new marker-discarding interpretation. The only difference comes in the case where $C := A \to B$:

$$T_C := \lambda r\, \lambda s\, T_\to \big(T_A(s_\llcorner)(r s_\lrcorner \boxed{\lrcorner})\big)\big(T_B((r \circ s_\llcorner)_\llcorner)(s_\lrcorner)\big).$$ □

Lemma 6.8 (Dialectica case disctinction with markers)**.** Let C be a formula in NA^ω and let $x : \rho^*(C)$ be a variable. Let D be a definition context associating a variable d_C with the term T_C defined in Lemma 6.7. Then there is a term $T^C_\bowtie : \rho^{\multimap}(C) \Rightarrow \rho^{\multimap}(C) \Rightarrow \rho^{\multimap}(C)$ with $\mathsf{FV}(T^C_\bowtie) \subseteq \mathsf{FV}(C) \cup \{x\}$, such that for $t_1, t_2 : \rho^{\multimap}(C)$ from the assumptions $u^{(i)} : (\!|C|\!)^x_{s_i} \to \mathrm{at}(m_i)$ we can prove

$$\begin{aligned} A_i :&\ (\!|C|\!)^x_s \to (\!|C|\!)^x_{s_i}, \\ B :&\ (\!|C|\!)^x_s \to \mathrm{at}(m), \end{aligned}$$

where $t_i := s_i \blacktriangleright m_i$ and $s \blacktriangleright m \overset{r}{=} t := D[T^C_\bowtie t_1 t_2]$ and $\lceil T^C_\bowtie \rceil$ is constant, not depending on the size of the formula C.

Proof. Using $T_\to$ from Lemma 1.51, let us define

$$\begin{aligned} T^C_\bowtie := \lambda y_1\, \lambda y_2\, &\textbf{let } s_1 := y_{1\llcorner} \textbf{ in let } m_1 := y_{1\lrcorner} \textbf{ in} \\ &\textbf{let } s_2 := y_{2\llcorner} \textbf{ in let } m_2 := y_{2\lrcorner} \textbf{ in} \\ &\mathsf{Cases}\, m_1 \big(\mathsf{Cases}\big(T_\to m_2 (d_C x s_1)\big)\big) y_2 (s_1 \blacktriangleright \mathsf{ff})\big) y_1. \end{aligned}$$

We will define proofs

$$\begin{aligned} &\mathcal{Q}^{(i)}_{m_1(,m_2)} : \vec{F} \to A_i, \ \text{ for } i = 1, 2, \\ &\mathcal{Q}_{m_1(,m_2)} : \vec{F} \to B, \qquad \text{where } F_j := (\!|C|\!)^x_{s_i} \to \mathrm{at}(m_i) \end{aligned}$$

for all possible values of the markers m_1 and m_2. Then we will be able to define

$$\mathcal{Q}^{(i)} := \mathcal{C}\, m_1\, (\mathcal{C}\, m_2\, \mathcal{Q}^{(i)}_{\mathsf{tt},\mathsf{tt}}\, \mathcal{Q}^{(i)}_{\mathsf{tt},\mathsf{ff}})\, \mathcal{Q}^{(i)}_{\mathsf{ff}}\, u'\, u'' \text{ for } i = 0, 1, 2.$$

We note that by definition

$$\begin{aligned} &t\,[m_1 := \mathsf{ff}] \overset{r}{=} t_1, \text{ and} && t\,[m_1 := \mathsf{tt}]\,[m_2 := \mathsf{ff}] \overset{r}{=} t_2, \text{ hence we can define}\\ &\mathcal{Q}_{\mathsf{ff}} := \lambda u'\,\lambda u''\,u', && \mathcal{Q}_{\mathsf{tt},\mathsf{ff}} := \lambda u'\,\lambda u''\,u'',\\ &\mathcal{Q}'_{\mathsf{ff}} := \lambda u'\,\lambda u''\,\lambda u\,u, && \mathcal{Q}''_{\mathsf{tt},\mathsf{ff}} := \lambda u'\,\lambda u''\,\lambda u\,u. \end{aligned}$$

For the rest of the cases we use the fact that the premise of $A_{3-i}\,[m_i := \mathsf{ff}]$ implies $(\!|C|\!)^x_{s_i}$, which contradicts the assumption $u^{(i)}\,[m_i := \mathsf{ff}]$. Hence, we define

$$\begin{aligned} \mathcal{Q}''_{\mathsf{ff}}, &:= \lambda u'\,\lambda u''\,\lambda u\,\mathsf{efq}(u'u),\\ \mathcal{Q}'_{\mathsf{tt},\mathsf{ff}} &:= \lambda u'\,\lambda u''\,\lambda u\,\mathsf{efq}(u''u). \end{aligned}$$

We are left only with the case where $m_1 = m_2 = \mathsf{tt}$. Note that

$$\begin{aligned} &T^C_{\bowtie}(s_1 \blacktriangleright \mathsf{tt})(s_2 \blacktriangleright \mathsf{tt}) \overset{r}{=} \mathsf{Cases}\,(T_C x s_1)\,(s_2 \blacktriangleright \mathsf{tt})\,(s_1 \blacktriangleright \mathsf{ff}), \text{ hence}\\ &A_i\,[m_1, m_2 := \mathsf{tt}] = (\!|C|\!)^x_{\mathsf{Cases}\,(T_C x s_1)\,s_2\,s_1} \to (\!|C|\!)^x_{s_i},\\ &B\,[m_1, m_2 := \mathsf{tt}] = (\!|C|\!)^x_{\mathsf{Cases}\,(T_C x s_1)\,s_2\,s_1} \to \mathrm{at}(T_C x s_1). \end{aligned}$$

Let $D_i := (\!|C|\!)^x_{s_i}$. Let us assume that we have proof terms $K : \mathrm{at}(T_C x s_1) \to (\!|C|\!)^x_{s_1}$ and $L : (\!|C|\!)^x_{s_1} \to \mathrm{at}(T_C x s_1)$. Similarly to Lemma 2.19 and Lemma 4.22, we define

$$\begin{aligned} \mathcal{Q}'_{\mathsf{tt},\mathsf{tt}} &:= \lambda u'\,\lambda u''\,\mathsf{CD}_{b,H_1}(T_C x s_1)(\lambda u^{\mathrm{at}(T_C x s_1)}\,\lambda v^{D_2}\,Ku)(\lambda u^{\mathrm{at}(T_C x s_1)\to\mathrm{F}}\,\lambda w^{D_1}\,w),\\ \mathcal{Q}''_{\mathsf{tt},\mathsf{tt}} &:= \lambda u'\,\lambda u''\,\mathsf{CD}_{b,H_2}(T_C x s_1)(\lambda u^{\mathrm{at}(T_C x s_1)}\,\lambda v^{D_2}\,v)(\lambda u^{\mathrm{at}(T_C x s_1)\to\mathrm{F}}\,\lambda w^{D_1}\,\mathsf{efq}_{D_2}(u(Lw))),\\ &\quad \text{for } H_i := (\!|C|\!)^x_{\mathsf{Cases}\,b\,s_2\,s_1} \to D_i. \end{aligned}$$

Finally, we set

$$\mathcal{Q}_{\mathsf{tt},\mathsf{tt}} := \lambda u'\,\lambda u''\,\mathsf{CD}_{b,H_0}(T_C x s_1)(\lambda u^{\mathrm{at}(T_C x s_1)}\,\lambda v^{D_2}\,\mathsf{AxT})(\lambda u^{\mathrm{at}(T_C x s_1)\to\mathrm{F}}\,\lambda w^{D_1}\,u(Lw)),$$

for $H_0 := (\!|C|\!)^x_{\mathsf{Cases}\,b\,s_2\,s_1} \to \mathrm{at}(b)$. □

We are ready to prove soundness of marked counterexamples. The proof will be a modification of the proof of Theorem 4.23. The only change will be in the treatment of challenges, where context-dependent *marked* negative witnessing terms $[\![M]\!]^-_i : \rho^{-\circ}(C_i)$ will be extracted.

Theorem 6.9 (Soundness of counterexample marking). *Let $A \in \mathrm{NA}^\omega$ be a formula and let $\mathcal{P}^A$ be a proof term with assumptions among $\{u_i : C_i\}_{i\geq 1}$. Let us have fresh witnessing variables $X = \{x_i : \rho^*(C_i)\}$, each one associated uniquely with an assumption variable u_i and let $y_A : \rho^-(A)$ be a fresh challenging variable associated uniquely with the formula A. Then there is a term $\{\!|\mathcal{P}|\!\}$ and proofs $\overline{\mathcal{P}} : (\!|A|\!)^{\{\!|\mathcal{P}|\!\}^+}_{y_A}$ and*

$\overline{\overline{\mathcal{P}}}_i : (\!|C_i|\!)^{x_i}_{s_i} \to \mathsf{at}(m_i)$, *where* $s_i \blacktriangleright m_i \overset{r}{=} \{\!|\mathcal{P}|\!\}^-_i y_A$ *and*

1. $\mathsf{FA}(\overline{\mathcal{P}}) \subseteq \{v_i : (\!|C_i|\!)^{x_i}_{s_i}\}$ *and* $\mathsf{FA}(\overline{\overline{\mathcal{P}}}_i) = \emptyset$,
2. $\mathsf{FV}(\{\!|\mathcal{P}|\!\}) \subseteq \mathsf{FV}(\mathcal{P}) \cup X$,
3. $\lceil \{\!|\mathcal{P}|\!\} \rceil \leq K(\lceil \mathcal{P} \rceil \llbracket \mathcal{P} \rrbracket^2)$ *for a fixed constant* K, *not depending on* $\mathcal{P}$.

Proof. *Case* u_1^A. We set as before $\llbracket \mathcal{P} \rrbracket := \lambda y_A\, []$, $\llbracket \mathcal{P} \rrbracket^+ := x_1 y_A$ and set $\llbracket \mathcal{P} \rrbracket^-_1 := y_A \blacktriangleright \mathsf{tt}$. Then $\{\!|\mathcal{P}|\!\}^+ \overset{r}{=} \lambda y_A\, x_1 y_A \overset{r}{=} x_1$ and $\{\!|\mathcal{P}|\!\}^-_i y_A \overset{r}{=} y_A \blacktriangleright \mathsf{tt}$, and as before we can define $\overline{\mathcal{P}} := v_1$. On the other hand, $m_1 \overset{r}{=} \mathsf{tt}$, hence $\overline{\overline{\mathcal{P}}}_1 := \lambda v\, \mathsf{AxT}$. The size bounds and the variable condition also hold as in Theorem 4.23.

Case $\lambda u_0^B\, M^C$. The extracted terms from Theorem 4.23 are still applicable. $\overline{\mathcal{P}}$ is defined as before and $\overline{\overline{\mathcal{P}}}_i := \overline{\overline{M}}_i \xi$ for $i \geq 1$ with $\xi := [x_0 := y_A \llcorner]\, [y_C := y_A \lrcorner]$.

Case $M_1^{C \to A} M_2^C$. Let us denote $B := C \to A$. The extracted terms are defined almost as in Theorem 4.23, with the slight change that before applying $\{\!|M_2|\!\}^-_i$ to $\{\!|M_2|\!\}^+ \lrcorner$, the marker needs to be discarded. The change is emphasized by a box below:

$$\llbracket \mathcal{P} \rrbracket^-_i := \llbracket M_1 \rrbracket^-_i \overset{C_i}{\bowtie} f(z \lrcorner \boxed{\lrcorner}) \triangleright i$$

The case distinction is altered to use the appropriate term from Lemma 6.8:

$$t_1 \overset{u_i}{\bowtie} t_2 := \begin{cases} t_1, & \text{if } t_1 \equiv t_2, \\ T^{C_i}_{\bowtie} t_1 t_2, & \text{otherwise.} \end{cases}$$

The proof $\overline{\mathcal{P}}$ is defined using $\mathcal{Q}'_i$ and $\mathcal{Q}''_i$ as before, and $\overline{\overline{\mathcal{P}}}_i := \mathcal{Q}_i(\overline{\overline{M}}_{1,i}\xi_1)(\overline{\overline{M}}_{2,i}\xi_2)$ with $\xi_{1,2}$ defined as in Theorem 4.23.

Cases $\lambda x^\rho\, M^B$ *and* $M^{\forall x^\rho\, B} t^\rho$. The proof of the same case in Theorem 4.23 still applies, because in both cases we neither remove nor introduce assumptions. In both cases $\overline{\overline{\mathcal{P}}}_i := \overline{\overline{M}}_i \xi$.

Case $\mathcal{C}^{b,A}\, b M_{\mathsf{tt}}^{A[b:=\mathsf{tt}]} M_{\mathsf{ff}}^{A[b:=\mathsf{ff}]}$. We define the extracted terms exactly as in Theorem 4.23 and set $\overline{\overline{\mathcal{P}}}_i := \mathcal{C}\, b\, \overline{\overline{M}}_{\mathsf{tt}}\, \overline{\overline{M}}_{\mathsf{ff}}$.

Case $\mathsf{Ind}_{\mathsf{N}}^{n,A}\, n\, M_1^{A[n:=0]}\, (\lambda n\, \lambda u_0^A\, M_2^{A[n:=\mathsf{S}n]})$. By induction hypothesis we have proofs

$$\begin{aligned} \overline{\overline{M}}_{1,i} &: (\!|C_i|\!)^{x_i}_{s_i} \to \mathsf{at}(m_i) \text{ for } i \geq 1, \\ \overline{\overline{M}}_{2,0} &: (\!|A|\!)^{x_0}_{r_0} \to \mathsf{at}(n_0) \text{ and} \\ \overline{\overline{M}}_{2,i} &: (\!|C_i|\!)^{x_i}_{r_i} \to \mathsf{at}(n_i) \text{ for } i \geq 1, \end{aligned}$$

where $s_i \blacktriangleright m_i \overset{r}{=} \{\!|M_1|\!\}^-_i y_A$ and $r_j \blacktriangleright n_j \overset{r}{=} \{\!|M_2|\!\}^-_j y_A$ for $i \geq 1$ and $j \geq 0$.

Extracted terms are defined almost as before, but using the modified case distinction $\bowtie$ according to Lemma 6.8 and discarding the marker of the negative content of the induction hypothesis, as shown below:

$$[\![L]\!] := \mathcal{R}_{\mathsf{N}}\, n\, \{\!|M_1|\!\}\big(\lambda n\, \lambda p\, \textbf{let } x_0 := p_{\llcorner} \textbf{ in } [\![M_2]\!][\textbf{let } z := p([\![M_2]\!]_0^-\boxed{\lrcorner}) \textbf{ in } [\,]]\big).$$

Let us denote $t_i \blacktriangleright p_i \overset{r}{=} \{\!|\mathcal{P}|\!\}_i^- y_A$. By definition we have

$$\begin{aligned}
(t_i \blacktriangleright p_i)\,[n := 0] \quad &\overset{r}{=} s_i \blacktriangleright m_i,\\
(t_i \blacktriangleright p_i)\,[n := \mathsf{S}n] &\overset{r}{=} \textbf{let } x_0 := \{\!|\mathcal{P}|\!\}^+ \textbf{ in } (r_i \blacktriangleright n_i) \overset{u_i}{\bowtie} (\{\!|\mathcal{P}|\!\}_i^- r_0)\\
&\overset{r}{=} \textbf{let } x_0 := \{\!|\mathcal{P}|\!\}^+ \textbf{ in } (r_i \blacktriangleright n_i) \overset{u_i}{\bowtie} (\textbf{let } y_A := r_0 \textbf{ in } (t_i \blacktriangleright p_i)).
\end{aligned}$$

We will define proofs $\tilde{\mathcal{P}}_i$ of

$$\forall y_A \left((\!|C_i|\!)_{t_i}^{x_i} \to \mathrm{at}(p_i) \right),$$

because then we can set $\overline{\overline{\mathcal{P}}}_i := \tilde{\mathcal{P}}_i y_A$. We use the proofs $\mathcal{Q}_i$ from Lemma 6.8 and define $\tilde{\mathcal{P}}_i$ by induction as follows:

$$\tilde{\mathcal{P}}_i := \mathsf{Ind}_{\mathsf{N}}\, n\, (\lambda y_A\, \overline{\overline{M}}_{1,i})\, \Big(\lambda n\, \lambda p\, \lambda y_A\, \mathcal{Q}_i\, (\overline{\overline{M}}_{2,i}\xi)\, \big(p(r_0\xi)\big)\Big),$$

where $\xi := \big[x_0 := \{\!|\mathcal{P}|\!\}^+\big]$.

Case $\mathsf{Ind}_{\mathsf{L}(\rho)}^{l,A}\, l\, M_1^{A[l:=\mathsf{nil}]}\, (\lambda x\, \lambda l\, \lambda u_0^A\, M_2^{A[l:=x\,::\,l]})$. We adopt all the definitions from the previous case and set

$$\tilde{\mathcal{P}}_i := \mathsf{Ind}_{\mathsf{L}(\rho)}\, l\, (\lambda y_A\, \overline{\overline{M}}_{1,i})\, \Big(\lambda x\, \lambda l\, \lambda p\, \lambda y_A\, \mathcal{Q}_i\, (\overline{\overline{M}}_{2,i}\xi)\, \big(p(r_0\xi)\big)\Big). \qquad \square$$

Corollary 6.10 (Extraction with marked counterexamples)**.** *Let $\mathcal{P} : C$ be a closed proof in* NA$^\omega$*. There is a closed term $\{\!|\mathcal{P}|\!\}^+ : \rho^*(C)$ with $\lceil \{\!|\mathcal{P}|\!\}^+ \rceil \leq K(\lceil \mathcal{P} \rceil [\![\mathcal{P}]\!]^2)$, and a proof*

$$\overline{\mathcal{P}} : \forall y^{\rho^-(C)}\, |C|_{(y^{-\blacktriangleright})^{\downarrow -}}^{(\lambda z\, (\{\!|\mathcal{P}|\!\}^+ z^{\blacktriangleleft -})^{+\blacktriangleright})^{\downarrow +}}.$$

Proof. Follows from Corollary 4.27, Proposition 6.6 and Theorem 6.9. $\square$

6.4 Infinite Pigeonhole Principle revisited

As already visible by the proof of Theorem 6.9, the counterexample marks do not increase the size of the extracted program significantly. In the case of the Infinite Pigeonhole Principle, we obtain the program shown in Figure 6.2. The challenges of four different assumptions are annotated with tt markers. They are discarded at some point by $\lrcorner$ projection, as shown by a box. We should note that from the four markers only two are relevant. These are the markers for w and for u_2, since they are the only assumptions over which a case distinction is needed. It is clear that every counterexample marker can be introduced independently for each assumption. Thus, we can mark only assumptions which participate in a case distinction. In this case study, the markers introduced in g_1 correspond to the assumptions u_1 and v_1 and are redundant.

Let us reason about the average time complexity of the program in 6.2 following the argument in Section 4.8.2. The major benefit from the counterexample marker comes in the case distinction $\overset{w}{\bowtie}$. As already discussed, in the worst case, the number of invocations of x_w in z_{14} would be $2n$ for every fixed r. However, because of the marker-aware definition of $\overset{w}{\bowtie}$, the function x_w will not be called if we have already found a counterexample n, such that $n \leq m$ and $fm \neq q$. Assuming an uniformly distributed random sequence f, we will find a counterexample with probability $\frac{r-1}{r}$. At first, this does not seem as a worthy improvement, since x_w is called anyway in $z_{5\llcorner}$ to construct the list. However, here a lazy strategy of evaluation is assumed, and since $p_{\llcorner}$ will no longer be needed after a counterexample is found, $z_{14\lrcorner}$ will be evaluated only after z_{15} has been already computed. Hence x_w will actually be invoked n times only after $z_{14\llcorner}$ has been correctly computed.

The extracted program with counterexample markers now behaves very similarly to the program obtained via refined A-translation in Section 3.3.2. For every colour $q < r$ the program calculates a sequence-extending function h_q, such that $h_q n \geq n$ and $f(h_q n) = q$. The function h_{q+1} is constructed by taking the counterexample index for the function h_q by computing $x_{u_2} \langle q, h_q \rangle$, as is done in z_7. The counterexample index is obtained by $z_{14\llcorner}$, i.e., by recursion on n, which terminates as soon as a counterexample is found. Thus, similarly to the program in 3.3.2, we have r recursions on n, one for each colour and we return undisturbed lists of indices, i.e., indices of colour q, between which there are only colours $\leq q$. With highest probability we will obtain indices of the largest colour $r-1$, which will execute n recursive calls in order to obtain n occurrences of $r-1$. Each of these recursive calls will compute an index of the colour $r-1$, which will be on average n indices after the last found occurrence of this colour. After the found counterexample is marked and because of the maximum operation in the function g_1, a new index (a "step forwards") is considered only in z_3, which is computed in a recursive step of z_{14}. Thus the number of considered indices closely follows the number of recursion steps executed by the program. Moreover, when an index of colour q is considered, $q-1$ recursion folds need to be carried out, since this index is a counterexample for all colours less than q. This is achieved by the function g_2. Hence, the consideration of every new index costs on average $r/2$ recursive steps. In total, we obtain that in the average case we have $O(nr^2)$ recursive steps in order to compute a list of n indices of colour $r-1$. Since lists of lower colours will be found even earlier, we can conclude that the average time complexity of the program is now $O(nr^2)$. In fact, the program is now extensionally equal to the program $[\![\mathcal{P}]\!]^\circ$, obtained via refined A-translation in Section 3.3.2, and both of the programs perform in the average case as good as the direct algorithm, given in Section 3.3.4.

$$
\lambda\langle r,f,n\rangle
\left(\begin{array}{l}
\textbf{let } z_{14} := \lambda\langle q,x_w\rangle \left(\begin{array}{l}
\textbf{let } z_8 := \lambda n\,\lambda p \left(\begin{array}{l}
\textbf{let } l := p_{\llcorner} \textbf{ in}\\
\textbf{let } z_5 := \left(\begin{array}{l} \mathcal{R}\, l\ \Box\ (\lambda x\,\lambda l\,\lambda x_{p'}\\ \textbf{let } z_3 := \mathsf{S}x \textbf{ in}\\ \langle x_w z_3 :: x :: l \blacktriangleright \mathsf{tt}, z_3 \blacktriangleright \mathsf{tt}\rangle)\end{array}\right) \textbf{ in}\\
\left\langle z_{5\llcorner}, p_{\lrcorner} \overset{w}{\bowtie} z_{5\lrcorner}\right\rangle
\end{array}\right) \textbf{ in}\\
\textbf{let } z_9 := \mathcal{R}\, n\, \langle x_w 0{:} \blacktriangleright \mathsf{tt}, 0 \blacktriangleright \mathsf{tt}\rangle\, z_8 \textbf{ in } \langle z_{9\lrcorner}, z_{9\llcorner}\rangle
\end{array}\right) \textbf{ in}\\
\textbf{let } z_{15} := \left(\begin{array}{l}
\mathcal{R}\, r\, (\lambda\langle f,x_{u_2}\rangle\,\Box)\Big(\lambda r\,\lambda x_p\,\lambda\langle f,x_{u_2}\rangle\\
\textbf{let } g_3 := \lambda n_1 \left(\begin{array}{l}
\textbf{let } g_1 := \lambda n_2\, \langle (n_1 \sqcup n_2) \blacktriangleright \mathsf{tt}, (n_1 \sqcup n_2) \blacktriangleright \mathsf{tt}\rangle \textbf{ in}\\
\textbf{let } g_2 := \lambda\langle q,x_w\rangle \left(\begin{array}{l} \textbf{let } z_2 := \langle q, \lambda n_2\, n_1 \sqcup x_w n_2\rangle \textbf{ in}\\ \langle x_{u_2} z_2, z_2 \blacktriangleright \mathsf{tt}\rangle \end{array}\right) \textbf{ in}\\
\textbf{let } z_4 := x_p\, \langle f\lceil n_1, g_{2\llcorner}\rangle \textbf{ in } \langle g_1(z_{4\lrcorner})_{\llcorner}, g_1(z_{4\lrcorner})_{\lrcorner}, g_2(z_{4\llcorner})_{\lrcorner}\rangle
\end{array}\right) \textbf{ in}\\
\textbf{let } z_5 := \langle r, g_{3\llcorner}\rangle \textbf{ in}\\
\textbf{let } z_7 := g_3(x_{u_2} z_5) \textbf{ in } \left\langle (z_5 \blacktriangleright \mathsf{tt}) \overset{u_2}{\bowtie} z_{7\lrcorner}, z_{7\llcorner}\right\rangle\Big)\,\langle f, z_{14\llcorner}\rangle
\end{array}\right) \textbf{ in}\\
\langle z_{15\lrcorner}, z_{14\lrcorner}(z_{15\llcorner}\boxed{\lrcorner})\rangle
\end{array}\right)
$$

where

$$
\begin{aligned}
(n_1 \blacktriangleright m_1) \overset{w}{\bowtie} (n_2 \blacktriangleright m_2) \equiv\ & \mathsf{Cases}\, m_1\, (\mathsf{Cases}\, m_2\, \textbf{let } m := x_w n_1 \textbf{ in}\\
& \mathsf{Cases}\, (T_{\to}(n_1 \le m)(fm = q))\, (n_2 \blacktriangleright m_2)\, (n_1 \blacktriangleright \mathsf{ff}))\, (n_1 \blacktriangleright m_1),\\
(\langle q_1,h_1\rangle \blacktriangleright m_1) \overset{u_2}{\bowtie} (\langle q_2,h_2\rangle \blacktriangleright m_2) \equiv\ & \mathsf{Cases}\, m_1\, (\mathsf{Cases}\, m_2\, \textbf{let } n := x_{u_2}\, \langle q_1,h_1\rangle \textbf{ in let } m := h_1 n \textbf{ in}\\
& \mathsf{Cases}\, (T_{\to}(n \le m)(fm \ne q_1)), (\langle q_2,h_2\rangle \blacktriangleright m_2)\, (\langle q_1,h_1\rangle \blacktriangleright \mathsf{ff}))\, (\langle q_1,h_1\rangle \blacktriangleright m_1)
\end{aligned}
$$

Figure 6.2: Program extracted from the Unbounded Pigeonhole Principle with counterexample markings

CONCLUSION

In this thesis we followed an empirical approach for comparing two different methods for obtaining functional programs from proofs in classical logic: the refined A-translation and Gödel's Dialectica interpretation. We expressed non-constructive proofs in an arithmetical system with higher types equipped with a restricted negative language for formulas. This choice made it possible to achieve a fair comparison by applying both interpretations to the same proof term.

Three case studies were selected to examine the behaviour of the two extraction methods. Stolzenberg's binary tape was chosen as a minimal example involving non-trivial use of classical reasoning. The obtained programs demonstrated that the two interpretations reflect the indirect reasoning by backtracking, which, however, is driven by different means: continuations and counterexamples. Both programs exhibited an asymmetry, which was discussed by many authors [Coq95, Mur91, BBS97, Urb00, Sei03, Mak06, Rat10] and is traced to the use of classical logic. The program obtained via the Dialectica interpretation was already unmanageable for such a simple example, which showed that there is definitely room for improvement.

The second example of finding an integer root of an unbounded function was chosen as an instance of the minimum principle, which is a convenient non-constructive tool for selecting witnesses from a non-empty well-ordered set. It demonstrated how programs extracted via the two methods are not necessarily extensionally equal, since in the Dialectica interpretation we have a freedom of choice in the case distinction operator. Moreover, there are different situations in which one of the choices is more efficient than the other. Our proposal is that instead of giving an *a priori* preference to one of the counterexample candidates, it is more suitable to introduce a boolean flag, which chooses the more efficient candidate during the evaluation of the program.

The last and most complex considered example is the Unbounded Pigeonhole Principle, derived as a simple inductive corollary of the Infinite Pigeonhole Principle. This case study was important, because it clearly separated the non-constructive argument from its constructive application. Both proof components involved induction and thus had a non-trivial computational meaning, making it possible to estimate their specific contribution to the complexity of the program. The extracted terms demonstrated a magnification of the asymmetry observed in Stolzenberg's example. The colours were considered in priority order, which made it possible to construct extreme counterexamples on which the number of evaluation steps is exponential in the number of colours. This is contrasted to the polynomial time complexity of a direct algorithm. However, the program obtained via refined A-translation showed a very interesting feature: on average, the number of performed steps is of the same polynomial order as the direct program! This result could advocate the practical value of extracting from non-constructive proofs: if we are willing to sacrifice worst-case complexity, we can obtain a program of the same average behaviour from a classical proof, which is easier to define than its constructive counterpart. Moreover, if we consider NP-hard problems, for which no polynomial algorithms are known, then we can only benefit from having a simpler proof, since we have no reasonable hope for a better performance if we used a constructive proof instead.

In the Infinite Pigeonhole Principle case study, the difference in readability and efficiency between the terms extracted with the two methods is most striking. The encouraging results yielded via refined A-translation naturally pose the question: can the same be achieved via the Dialectica interpretation? This question is answered positively throughout the rest of the text by gradual refinement of the interpretation aimed at obtaining better extracted terms.

The most obvious drawback of the Dialectica interpretation are the bloated programs, which are obtained by a direct application of the soundness theorem. Already Hernest and Kohlenbach noticed that special care needs to be taken about extracted terms in order to control their size [HK05]. The unreadable programs are not such a problem when the interpretation is used for manual "proof mining", since an expert would naturally simplify the terms by performing suitable sound ad-hoc reductions. However, when the method is applied completely automatically, it becomes vital to have means for systematic simplifications of the extracted term.

We noticed that the main reason for the large size of the programs is due to the use of substitution on the meta level, which leads to repetition of equal expressions in the term and also raises an issue of efficiency. The underlying cause for this repetition is the dual nature of Dialectica, in which every component of the proof has a positive and a negative reflection: as a witness and as a challenge. The proposed variant of the interpretation adjusted its syntactic representation, so that the common expressions are bound by a single context, in which positive and negative computational content is computed simultaneously. This mechanical factorisation is not necessarily reflected during evaluation — if we factor out an expression of non-ground type, we might need to reevaluate it when it is used simultaneously as a witness and as a challenge. This is unavoidable, as it constitutes an important part of the backtracking process, which is the computational footprint of the use of classical logic. In such cases the factorisation only reduces the size of the extracted term without having a positive or a negative effect on complexity. We can observe the beneficial effect of the reformulation only for the case of terms of ground type, which are reduced to a value only once before being used further.

The factorisation in extracted terms is in some sense overzealous, since it attempts to capture all possible sources of repetition. This calls for additional cleaning of the program to make it more readable. By executing only affine reductions we guarantee that the terms can only shrink, while the advantage of factorisation is preserved.

The obtained bound on the size of the programs was not completely linear, and this was due to our aim to express the terms completely in the simple language of the system. The overhead signified by the square of the maximal sequent length is caused by the need to pack and unpack positive and negative computations. We should be able to regain the linear bound if we extend the term language to include a form of pointers, which have constant access time, so that we do not need repeated projections to access different components. Nevertheless, even with the current formulation, the overhead is not that large: even for the most complex case study of the Infinite Pigeonhole Principle, the maximal sequent length equals to 6, which is quite small compared to the size of the whole proof. Naturally, extreme examples can always be built, such that the maximal sequent length is of the same order as the size of the proof. However, in the author's opinion, this overhead should not be of serious concern for practical use.

Another class of redundancies that can appear in an extracted term are irrelevant computations. When we consider proof interpretations separating the computational and logical components from the proof based on a syntactic criterion, it is possible that purely logical parameters are treated as computational. The uniform annotations present an additional syntax allowing for a finer separation based not only on the shape of the formula involved, but also on the specific use of the component in the proof. Being a dual extension of modified realisability, the possibilities for uniform annotations in the Dialectica interpretation are strictly larger than the original uniform quantifiers, suggested by Berger in [Ber05]. Although the theoretically possible uniform annotations for Dialectica are quite many (cf. [Tri09]), in this work we have restricted ourselves to those combinations which have some definite application. Hernest's first adaptation of Berger's uniform universal quantifier [Her07b] is completely sufficient for our case studies, but the examples in Chapter 5 demonstrate several cases in which other uniform annotations can be used. Our choice of considered combinations was motivated by the ability to express the modified realisability interpretation extended with Berger's uniform quantifiers using the Dialectica interpretation for the system $\overline{\mathrm{NA}^{\omega}}$.

An important topic for future research would be an algorithm for automatically inserting a maximal amount of uniform annotations, so that the proof remains computationally correct. Thus we will be able to automatically remove all redundant computations without the need to structurally modify the proof. It is clear that such a procedure must exist, since the proof is a finite object. A similar algorithm for modified realisability was already demonstrated by Ratiu and Schwichtenberg in [RS09]. An adaptation of the algorithm would probably be applicable to the more complicated system $\overline{\mathrm{NA}^{\omega}}$.

As was the case of common term factorisation, uniform annotations also have a twofold advantage. On one side, they remove terms, which are not needed to compute the final result. However, they do not necessarily remove only unreachable code; as demonstrated by Berger in [Ber05], we can remove slow computations, which are irrelevant and thus improve the worst time complexity of the program. It is important to note that the *time complexity* will only be improved when using an eager evaluation strategy: a lazy strategy would never evaluate irrelevant code. Nevertheless, such subterms would still exist in the program and removing them will be beneficial for the *space complexity* when we use call-by-name evaluation. The case studies considered here took advantage only of the cleaning aspect; the time complexity was not altered by the removal of irrelevant computations. An interesting topic of future research would be to compare the cleaning performed by a maximal uniform annotation of a proof with a purely computer-scientific approach of removing irrelevant function parameters, such as the work by Alpuente et al. [AEL02].

The results presented in the final chapter of the thesis allow for reducing the average time complexity of the program extracted from the Infinite Pigeonhole Principle from exponential to polynomial. However, this effect would not be as strong if counterexample marking was not applied in the context of the quasi-linear Dialectica interpretation. The reason is that if every step of the recursive computation of counterexamples was invoking a recursive computation of witnesses, as this is done in the original interpretation, then the otherwise linear process would turn into quadratic.

The optimisations of the Dialectica extraction process, which were presented in the last three chapters, will be implemented in the interactive proof assistant Minlog and would make it possible to extract shorter and more efficient programs. The practical benefit from these results would pave the way to considering more complicated case studies for extraction from non-constructive proofs. Possible examples would include classical proofs of existence, finding witnesses for which is NP-hard, such as Ramsey's theorem.

INDEX

LIST OF FIGURES

BIBLIOGRAPHY

[AEL02] M. Alpuente, S. Escobar, and S. Lucas. Removing redundant arguments of functions. *Algebraic Methodology and Software Technology*, pages 241–242, 2002.

[AF98] J. Avigad and S. Feferman. Gödel's functional ('Dialectica') interpretation. In Samuel Buss, editor, *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, pages 337–405. Elsevier, 1998.

[BB93] Franco Barbanera and Stefano Berardi. Extracting constructive content from classical logic via control-like reductions. In Marc Bezem and Jan Friso Groote, editors, *TLCA*, volume 664 of *Lecture Notes in Computer Science*, pages 45–59. Springer, 1993.

[BBS97] F. Barbanera, S. Berardi, and M. Schivalocchi. Classical programming-with-proofs in λ_{PA}^{Sym}: An analysis of non-confluence. *Lecture notes in computer science*, pages 365–390, 1997.

[BBS02] Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined program extraction form classical proofs. *Ann. Pure Appl. Logic*, 114(1–3):3–25, 2002.

[Ber95] Ulrich Berger. Programs from classical proofs. In *Proceedings of the 2nd Gauss Symposium: Munich, Germany, August 2-7, 1993. Mathematics and theoretical physics*, page 187. Walter De Gruyter Inc, 1995.

[Ber05] Ulrich Berger. Uniform Heyting Arithmetic. *Ann. Pure Appl. Logic*, 133(1–3):125–148, 2005.

[BES98] Ulrich Berger, Matthias Eberl, and Helmut Schwichtenberg. Normalisation by evaluation. In Bernhard Möller and J. V. Tucker, editors, *Prospects for Hardware Foundations*, volume 1546 of *Lecture Notes in Computer Science*, pages 117–137. Springer, 1998.

[BES03] Ulrich Berger, Matthias Eberl, and Helmut Schwichtenberg. Term rewriting for normalization by evaluation. *Inf. Comput.*, 183(1):19–42, 2003.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, New York, NY, USA, 1998.

[BNS00] Stephen J. Bellantoni, Karl-Heinz Niggl, and Helmut Schwichtenberg. Higher type recursion, ramification and polynomial time. *Ann. Pure Appl. Logic*, 104(1–3):17–30, 2000.

[BS95] Ulrich Berger and Helmut Schwichtenberg. Program extraction from classical proofs. In D. Leivant, editor, *Logic and Computational Complexity International Workshop LCC'94*, volume 960 of *Lectures Notes in Computer Science*, pages 177–194. Springer-Verlag, 1995.

[Coq94] Thierry Coquand. An analysis of ramsey's theorem. *Information and Computation*, 110(2):297–304, 1994.

[Coq95] Thierry Coquand. A semantics of evidence for classical arithmetic. *Journal of Symbolic Logic*, 60(1):325–337, 1995.

[DF92] Olivier Danvy and Andrzej Filinski. Representing control: A study of the CPS transformation. *Mathematical Structures in Computer Science*, 2(4):361–391, 1992.

[Dra80] A. Dragalin. New kinds of realisability and the Markov rule. *Dokl. Akad. Nauk. SSSR*, 251:534–537, 1980. in Russian.

[FF89] Matthias Felleisen and Daniel P. Friedman. A syntactic theory of sequential state. *Theor. Comput. Sci.*, 69(3):243–287, 1989.

[Fri78] Harvey Friedman. Classically and intuitionistically provably recursive functions. *Lecture Notes in Mathematics*, 669:21–27, 1978.

[GK10] Jaime Gaspar and Ulrich Kohlenbach. On tao's "finitary" infinite pigeonhole principle. *Journal of Symbolic Logic*, 75(1):355–371, 2010.

[Göd58] Kurt Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958.

[Gri74] V. N. Grishin. A nonstandard logic and its application to set theory. *Studies in Formalized Languages and Nonclassical Logics*, pages 135–171, 1974. In Russian.

[Gri81] V. N. Grishin. Predicate and set-theoretic calculi based on logic without contraction rules. *Izvestiya Akademii Nauk SSSR Seriya Matematicheskaya*, 45(1):47–68, 1981. In Russian.

[Gri90] Timothy Griffin. A formulae-as-types notion of control. In *POPL*, pages 47–58, 1990.

[Her07a] Mircea-Dan Hernest. Light dialectica program extraction from a classical fibonacci proof. *Electr. Notes Theor. Comput. Sci.*, 171(3):43–53, 2007.

[Her07b] Mircea-Dan Hernest. *Optimized programs from (non-constructive) proofs by the light (monotone) Dialectica interpretation.* PhD thesis, Ecole Polytechnique, 2007.

[HK05] Mircea-Dan Hernest and Ulrich Kohlenbach. A complexity analysis of functional interpretations. *Theor. Comput. Sci.*, 338(1–3):200–246, 2005.

[HO08] Mircea-Dan Hernest and Paulo Oliva. Hybrid functional interpretations. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *CiE*, volume 5028 of *Lecture Notes in Computer Science*, pages 251–260. Springer, 2008.

[HT10] Mircea-Dan Hernest and Trifon Trifonov. Light dialectica revisited. *Annals of Pure and Applied Logic*, 161(11):1313–1430, August 2010.

[Joh37] I. Johansson. Der Minimalkalkül, ein reduzierter intuitionistischer Formalismus. *Compositio Mathematia*, 4:119–136, 1937.

[Jø01] Klaus Frovin Jørgensen. Finite type arithmetic. Master's thesis, University of Roskilde, 2001.

[Kle45] Stephen C Kleene. On the interpretation of intuitionistic number theory. *The Journal of Symbolic Logic*, 10(4):109–124, December 1945.

[Kre51] Georg Kreisel. On the interpretation of non-finitist proofs, part i. *Journal of Symbolic Logic*, 16:241–267, 1951.

[Kre52] Georg Kreisel. On the interpretation of non-finitist proofs, part ii. *Journal of Symbolic Logic*, 17:43–58, 1952.

[Kre59] G. Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In A. Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North-Holland Publishing Company, 1959.

[Kre62] G. Kreisel. On weak completeness of intuitionistic predicate logic. *The Journal of Symbolic Logic*, 27(2):139–158, 1962.

[Kri04] Jean-Louis Krivine. Realizability in classical logic. *Panoramas et synthèses, Société Mathématique de France*, 2004.

[Lei75] Daniel Leivant. Strong normalization for arithmetic. In A. Dold and B. Eckmann, editors, *ISILC Proof Theory Symposion*, volume 500 of *Lecture Notes in Mathematics*, pages 182–197. Springer Berlin / Heidelberg, 1975.

[Lei85] Daniel Leivant. Syntactic translations and provably recursive functions. *Journal of Symbolic Logic*, 50(3):682–688, 1985.

[Lei01] D. Leivant. Termination proofs and complexity certification. In *Theoretical aspects of computer software*, pages 183–200. Springer, 2001.

[Mak06] Yevgeniy Makarov. *Practical program extraction from classical proofs*. PhD thesis, Indiana University, September 2006.

[mob] Mobius project — mobility, ubiquity and security: Enabling proof-carrying code for java on mobile devices. http://mobius.inria.fr/.

[Mur91] Chetan R. Murthy. Classical proofs as programs: How, what, and why. In J. Paul Myers Jr. and Michael J. O'Donnell, editors, *Constructivity in Computer Science*, volume 613 of *Lecture Notes in Computer Science*, pages 71–88. Springer, 1991.

[Nel47] D. Nelson. Recursive functions and intuitionistic number theory. *Transactions of the American Mathematical Society*, 61(2):307–368, 1947.

[NL96] George C. Necula and Peter Lee. Safe kernel extensions without run-time checking. In *OSDI*, pages 229–243, 1996.

[Oli08] P. Oliva. An analysis of Gödel's Dialectica interpretation via linear logic. *Dialectica*, 62(2):269–290, 2008.

[OW05] G.E. Ostrin and S.S. Wainer. Elementary arithmetic. *Annals of Pure and Applied Logic*, 133(1–3):275–292, 2005.

[Par92] Michel Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In Andrei Voronkov, editor, *LPAR*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 1992.

[Pra71] Dag Prawitz. Ideas and results of proof-theory. In J. E. Fenstad, editor, *Proceeding of the Second Scandinavian logic symposium*, pages 235–307, Amsterdam, 1971. North-Holland Pub. Co.

[Rat10] Diana Ratiu. *Refinement of Programs Extracted from Classical Proofs*. PhD thesis, Ludwig-Maximilian Universität — München, 2010. In progress.

[Ros53] G.F. Rose. Propositional calculus and realizability. *Transactions of the American Mathematical Society*, 75(1):1–19, 1953.

[RS09] Diana Ratiu and Helmut Schwichtenberg. Decorating proofs. To appear in Mints Festschrift, draft at: http://www.math.lmu.de/~schwicht/papers/mints09/deco20090728.pdf, 2009.

[RT09] Diana Ratiu and Trifon Trifonov. Exploring the computational content of the infinite pigeonhole principle. Draft at http://www.math.lmu.de/~trifonov/papers/iph.pdf, 2009. To appear in Proceedings of CiE 2008, Journal of Logic and Computation.

[Sch08] Helmut Schwichtenberg. Dialectica interpretation of well-founded induction. *Mathematical Logic Quarterly*, 54(3):229–239, 2008.

[Sei03] Monika Seisenberger. *On the Constructive Content of Proofs*. PhD thesis, Ludwig-Maximilian Universität — München, 2003.

[Spe62] Clifford Spector. Provably recursive functionals of analysis: a consistency proof of analysis by an ex- tension of principles in current intuitionistic mathematics. In F. D. E. Dekker, editor, *Recursive Function Theory: Proc. Symposia in Pure Mathematics*, volume 5, pages 1–27, Providence, Rhode Island, 1962. American Mathematical Society.

[SW10] Helmut Schwichtenberg and Stanley Wainer. Proof and computations. To appear, 2010.

[Tao07] Terence Tao. Soft analysis, hard analysis, and the finite convergence principle. http://terrytao.wordpress.com/2007/05/23/soft-analysis-hard-analysis-and-the-finite-convergence-principle/, May 2007.

[Tri09] Trifon Trifonov. Dialectica interpretation with fine computational control. In Klaus Ambos-Spies, Benedikt Löwe, and Wolfgang Merkle, editors, *Mathematical Theory and Computational Practice*, volume 5635 of *LNCS*, pages 467–477. Springer Berlin/Heidelberg, 2009. Proceedings of 5th Conference on Computability in Europe, CiE 2009, Heidelberg, Germany, July 19-24, 2009.

[Tri10a] Trifon Trifonov. Dialectica interpretation with marked counterexamples. In Steffen van Bakel, Stefano Berardi, and Ulrich Berger, editors, *CL&C*

2010, pages 86–98. MFCS & CSL, 2010. Workshop in Honour of Helmut Schwichenberg.

[Tri10b] Trifon Trifonov. Quasi-linear dialectica extraction. In Fernando Ferreira, Benedikt Löwe, Elvira Mayordomo, and Luís Mendes Gomes, editors, *CiE*, volume 6158 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 2010.

[Tro73] A. S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer-Verlag, 1973.

[TS00] Anne S. Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Number 43 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 2000.

[Urb00] Christian Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, October 2000.

[VB93] Wim Veldman and Marc Bezem. Ramsey's theorem and the pigeonhole principle in intuitionistic mathematics. *Journal of the Londn Mathematical Society*, s2–47(2):193–211, April 1993.