https://doi.org/10.20378/irbo-52663

A Modal Logic for Handling Behavioural Constraints in Formal Hardware Verification

Michael V. Mendler

Doctor of Philosophy
University of Edinburgh
1992

Abstract

The application of formal methods to the design of correct computer hardware depends crucially on the use of abstraction mechanisms to partition the synthesis and verification task into tractable pieces. Unfortunately however, behavioural abstractions are genuine mathematical abstractions only up to behavioural *constraints*, *i.e.* under certain restrictions imposed on the device's environment. Timing constraints on input signals form an important class of such restrictions. Hardware components that behave properly only under such constraints satisfy their abstract specifications only approximately. This is an impediment to the naive approach to formal verification since the question of how to apply a theorem prover when one only knows *approximately* what formula to prove has not as yet been dealt with.

In this thesis we propose, as a solution, to interpret the notion of 'correctness up to constraint' as a modality of intuitionistic predicate logic so as to remove constraints from the specification and to make them part of its proof. This provides for an 'approximate' verification of abstract specifications and yet does not compromise the rigour of the argument since a realizability semantics can be used to extract the constraints. Also, the abstract verification is separated from constraint analysis which in turn may be delayed arbitrarily. In the proposed framework constraint analysis comes down to proof analysis and a computational semantics on proofs may be used to manipulate and simplify constraints.

Acknowledgements

The development of this work has been influenced by discussions with my supervisors Rod Burstall and Mike Fourman, and my friend Terry Stroup. I am indebted to both Terry and Rod for their continued moral support and scientific guidance.

I would like to thank Bernhard Steffen and Eugenio Moggi for a pleasurable start in Edinburgh and for helping me along during my early days as a research student. Bernhard and Eugenio put me up in their flat at Mayfield Terrace after I had broken my collarbone. They looked after me for a fourtnight when I was hardly able to move.

I have benefited from discussions with Barry Jay, Randy Pollack, Zhaohui Luo, and James McKinna at Edinburgh, and Wolfgang Degen at Erlangen. At Erlangen the *Dienstagsclub*, most notably Norbert Götz, Uwe Nestmann, Thomas Amrhein, Martin Hofmann, and Martin Steffen, provided a stimulating environment and a lot of encouragement.

I am grateful to Klaus Müller-Glaser for his encouragement and the opportunity to become research assistant with the *Lehrstuhl für Rechnergestützten Schaltungsentwurf* at Erlangen where a fair part of this work was written. During the final phases of writing up I was employed by the *Lehrstuhl für Rechnerarchitektur und Verkehrstheorie* on the research project *Spezifikation und Verifikation verteilter Systeme.*

Norbert Götz, Matt Fairtlough, Zhaohui Luo, Stuart Anderson, and Georg Schied have read parts of this thesis in a draft version. My thanks go to them

for their valuable criticism and various suggestions for improvement. Also, I would like to thank Julian Bradfield for his assistance with type-setting and for supplying TEX-macros for proof trees. I owe special thanks to Kees Goossens who kindly helped with the binding and submission of this thesis.

This work was supported by scholarships from the *Studienstiftung des Deutschen Volkes* and the Stevenson Foundation, partly under SERC grant GR/F 35890 "Formal System Design", and by the *Deutsche Forschungsgemeinschaft*, SFB 182.

I am greatly indebted to my wife Marion for her love and for reassuring me all along. I will never forget the wonderful time we spent together in Scotland. During those few years of postgraduate study Marion gave birth to our girls Verena, Marie, and Sophie. I know she took over much of the strenuous and tiresome parts of caring for the babies, which took a lot of patience and understanding on her side.

Declaration

This thesis has been composed by myself. The work reported herein, except where indicated in the text, is my own, and has not been presented for any university degree before. Some of the introductory material in Chapter 1 and Section 2.3, and the example in Section 4.2 have already appeared in an early version as [Men91a].

Michael V. Mendler

Table of Contents

1 Introduction 1

1.1 Hardware Verification and Behavioural Abstraction 2

1.2 The Problem of Constraints 3

1.3 Aim of Thesis 5

1.4 Outline of Thesis 7

2 Motivation 9

2.1 Example 1 9

2.2 Example 2 11

2.3 Example 3 13

2.3.1 Synchronous Circuits 13

2.3.2 A Simple Circuit Design 15

3 Lax Logic 23

3.1 Definition of Lax Logic 28

3.1.1 Base Logic 28

3.1.2 Extending the Base Logic 35

3.1.3 Lax Logic 48

3.2 Constraint Extraction 59

3.2.1 Application of Constraint Extraction 71

3.3 Natural Deduction Proofs and Constraint Extraction 80

4 Application Examples 87

4.1 Decrementor, Incrementor, and Factorial 87
4.1.1 Decrementor 89
4.1.2 Composing Incrementor and Decrementor 96
4.1.3 Incrementor 98
4.1.4 Factorial 112
4.2 Example of Synchronous Hardware Design 128

5 Some Meta-Theory for Lax Logic **140**
5.1 On Kripke Semantics for $\Diamond$ 140
5.2 A Category Theoretical Interpretation of Lax Logic 152
5.2.1 Base Logic as an Indexed Preorder 152
5.2.2 Lax Logic as Indexed Category 166

6 Related Work **212**

7 Conclusion **222**
7.1 Further Research 224
7.1.1 Implementation 224
7.1.2 Application Areas 225
7.1.3 Meta-Theory 226
7.2 Open Problems 229

List of Figures

2-1 Xor-Gate and Level-Triggered Latch 16

2-2 Exclusive-Or and One-Unit Delay 17

2-3 Implementation of the *Modulo*-2 Counter 20

3-1 Well-Formed Terms of the Base Logic, I 31

3-2 Sequent Rules of the Base Logic 32

3-3 Derived Rules for the Base Logic 34

3-4 Well-Formed Terms of $\mathcal{B}$, II 38

3-5 Equality Axioms of $\mathcal{B}$ for Standard Data Types 40

3-6 Well-Formed Formulae of Lax Logic 50

3-7 Structural Rules of Lax Logic 52

3-8 Induction Rules of Lax Logic 52

3-9 Logical Inference Rules of Lax Logic 58

3-10 Constraint Extraction for Structural Rules of Lax Logic 65

3-11 Constraint Extraction for Logical Rules of Lax Logic, I 66

3-12 Constraint Extraction for Logical Rules of Lax Logic, II 68

3-13 Derivation of $\Diamond L$ from $\Diamond F, \Diamond M$ and Extracted Constraint Term . 69

3-14 A Simple Application of Constraint Extraction 77

3-15 Natural Deduction Rules of Lax Logic 85

3-16 Translation of Natural Deduction Trees into Constraint Terms . . 86

4-1 Derivation which Verifies the Decrementor Function 91

4-2 Derivation for Composition of Incrementor and Decrementor . . . 96

4-3 w-Bit Realization of Successor 102

4–4 Global Structure of Derivation 1 103
4–5 Sub-Derivation (P_1) 104
4–6 Step Case of Induction, Sub-Derivation (P_2) 105
4–7 Proof of Derivation 2 118
4–8 Proof of Derivation 3 119
4–9 Derivation for Abstracting *xor* as a Synchronous Device 131
4–10 Verification of Correctness for *Modulo*-2 Counter 134
4–11 Low-Level Implementation of *Modulo*-2 Counter 138

5–1 Interpretation of Structural Rules for $\mathcal{L}_0$ 204
5–2 Interpretation of Logical Rules for $\mathcal{L}_0$ 205

Chapter 1

Introduction

This introductory chapter, as a motivation for the results to be described in this thesis, identifies a prominent practical problem arising in the application of interactive theorem proving to the formal synthesis of correct computer hardware which so far has no satisfactory solution.

The point of departure of this thesis is the notion of a behavioural constraint and its specific nature. In this chapter this notion, as understood here, is defined and discriminated from other interpretations of the term. Further, the practical problem which results from the need to handle constraints in verifying circuits across incomplete abstractions is explained in some detail. The adequate formalization of constraints, so it is argued, calls for special mechanisms to be introduced in a theorem prover. Finally, the goal of our research is formulated, the results of which are summed up in this thesis.

At the end of this chapter a short summary of the results and the structure of the thesis is given.

1.1 Hardware Verification and Behavioural Abstraction

The application of interactive theorem proving to the design of computer hardware is taking first steps from pure correctness analysis to interactive synthesis. Examples of such design tools, which were only recently developed, are LAMBDA [FM89] and VERITAS [HDL89]. Both systems, at their roots interactive theorem provers, directly aim at providing an environment for the synthesis of correct digital circuits by stepwise refinement of an abstract behavioural specification. The guiding idea underlying such design tools is to take established engineering techniques in the practical design of hardware and gradually to formalize them in terms of mathematical logic.

Hardware design proceeds — ignoring false starts and similar matters — by refining numerous levels of abstraction, beginning typically with architectural level block diagrams and ending, by way of register-transfer and gate networks, in a transistor layout that is implemented in a physical medium. This process postpones detailed design decisions until they are appropriate and factors the verification of the final implementation's correctness into a sequence of smaller steps: If at each level of abstraction the design is shown to behave as required by the next higher level, then the final implementation meets its original specification.

Implementing this technique in terms of formal logics on a theorem prover presupposes rigorous mathematical models for the descriptions at each level of abstraction as well as corresponding abstraction and realization functions, so that behaviours described at different levels may be compared. The spectrum of models ranges from discrete computational structures appropriate to algorithmic descriptions down to differential equations modelling electrical behaviour of transistors.

Behavioural abstractions by their very nature are genuine mathematical abstractions only up to behavioural constraints, *i.e.* under certain restrictions im-

posed on the device's environment. Examples of such constraints are timing constraints for flipflops or synchronous circuits, handshaking constraints for speed-independent circuits, or the requirement that input data be within a specified integer range in order to avoid overflow of arithmetical operations. Hence, in general, the synthesis of a hardware design through levels of abstraction can be verified only up to certain constraints, and the question arises how this should best be implemented in modern interactive theorem provers.

1.2 The Problem of Constraints

A typical phenomenon which one encounters with the implementation of even conceptually simple abstraction steps which are standard practice in hardware engineering is that they cannot be formalized without introducing constraints. Now, the notion of constraint in computer science and in particular in hardware engineering is heavily overloaded, so a definition of what is meant by it in this thesis is in order:

> A *constraint* is a *restriction on the environment of a (hardware or software) component under which a particular abstraction of its behaviour is valid.*

As an example consider the passage from a sequential circuit, built according to the synchronous design paradigm, to its abstract description in terms of a finite state machine. Here the abstraction is only valid as long as the environment (among other things) obeys *setup* and *hold* timing constraints which require that all input lines of the sequential circuit must be kept stable during a certain well-defined phase of the clock. Clearly, the necessity for imposing timing constraints is a general phenomenon, not restricted to the synchronous case. It is an even more important issue in asynchronous designs. [Her88a,Ung69,Sub88a].

Although it is generally recognized that constraints are an essential concept in hardware design the specific nature of constraints and the question of whether

constraints are amenable to or require special treatment in a hardware theorem prover has not been adequately discussed. The work presented in this thesis is motivated by the following four problems associated with having to handle constraints on a theorem prover.

- By definition a constraint is the price one has to pay for making a particular behavioural abstraction work, *i.e.* it embodies an unwanted byproduct of abstraction. Therefore, in contrast to specifications, constraints should ideally be suppressible for a first cut of a design. This is the way engineers proceed but it is not at all obvious how such a scheme can be formalized on a theorem-prover. The question of how to apply a theorem prover when one only knows approximately what formula to prove has not as yet been dealt with. After all, in formal reasoning by its very nature, there is no room for 'rough estimates' or 'approximate specifications'. One is forced to cross the t's and dot the i's and cannot leave out anything on which correctness of an abstraction depends.

- Another good reason for distinguishing constraints from specifications is due to the fact that the former are conditions on the environment of a component whereas the latter is a condition on the component itself. Consequently, in the design process, constraints are being accumulated bottom-up as more and more parts of a circuit become implemented while at the same time the specification is being resolved top-down. This again requires special effort on a theorem-prover as it amounts to some kind of 'bidirectionality' in proof steps. At each intermediate design state and abstraction level information about the verification goal is incomplete: one does not know how to weaken the abstract specification to accommodate for potential input constraints until the final implementation has been given.

- The interaction between abstraction and constraints poses a tangled problem. Constraints interfere with the essential idea of reasoning about a behaviour in *abstract* terms which is to avoid details specific to the implementation at the more *concrete* level. For it is impossible to work with the device's abstract behaviour without at the same time having to deal with the concrete-level con-

straints on which it depends. To verify, for instance, that the behaviour of a composite device meets its abstract specification it does not suffice simply to compose the abstract specifications of its components. The verification also has to show that at the concrete level the composition does not violate the constraints of each component (which in general, will make it necessary to impose constraints again on the environment of the composite device).

• Constraints defeat the idea of top-down refinement, which is first to decompose a system into components at the abstract level and then independently to implement each component at the concrete level; Verifying constraints requires knowledge both of the overall structure of the system (the environment of a component) from the abstract level *and* of the implementation (the constraints of a component) at the lower level. In short, the general situation in the modelling of hardware seems to be that of incomplete abstractions, *i.e.* abstractions *modulo constraints*. The constraints on which the abstraction depends embody residual aspects of the concrete level that impinge on the subsequent design and cannot be abstracted away once and for all.

1.3 Aim of Thesis

Recognizing the special nature of constraints as opposed to specifications and their importance for the design process, this thesis investigates the advantages of distinguishing at the level of the logical inference mechanism between both kinds of propositions; namely those pertaining to constraints and those pertaining to specifications. This thesis contributes towards a systematic method for handling constraints in logic-based design tools which is tailored to the specific nature of constraints.

We want to make *reasoning about abstract behaviour* and *constraint analysis* fall into two separate verification passes, rather than having them intertwined as the straight-forward approach suggests. This thesis introduces and justifies an extension to ordinary predicate logic in which the main verification of an abstract

behaviour is truly an abstract verification in that it does not have to be concerned with constraints. It proceeds by assuming a successful constraint analysis wherever it depends on constraints. In the course of this main verification information about the constraints is accumulated as a proof obligation to be filled in at a later stage. Ideally (in an implementation of this logic), the remaining verification task corresponding to constraint analysis would then be handed over to a specialized tool or proof tactic. In some cases it could be solved automatically, for instance extracting the minimal clock period for a synchronous system. In other cases, where the logic is undecidable, it has to be done interactively. An example of this would be proving that the output of a certain integer function lies within a given finite range.

In this thesis we propose a notion of *lax proof* and *lax specification* which provides for 'approximate' verification of abstract specifications, and for a systematic method of removing from the verification engineer the burden of having to handle constraints. In order not to compromise the rigour of formal proofs this degree of looseness is implemented within the framework of a strictly deductive logic.

More concretely, we interpret the notion of 'correctness up to constraint' as a modality of intuitionistic predicate logic so as to remove constraints from specifications and to make them part of their proofs. This provides for an 'approximate' verification of abstract specifications, and yet it does not compromise the rigour of the argument since a realizability semantics can be used to extract the constraints. Thereby the task of verifying an abstract behaviour is separated from the task of analyzing constraints which may be delayed arbitrarily. Thus, in the proposed framework constraint analysis comes down to proof analysis and a computational semantics on proofs may be used to manipulate and simplify constraints.

The idea leading to *Lax Logic* presented in this thesis reflects good engineering practice: In a first approximation one tries to establish the feasibility of a design.

Only then is it worthwhile to attempt a complete validation in a second step. Lax Logic is an attempt to formalize this engineering principle mathematically and to implement it at the root of a formal predicate logic.

1.4 Outline of Thesis

We begin in Chapter 2 with motivating the problem of constraints on a characteristic example from hardware verification.

The formal calculus of *Lax Logic* will be set up in Section 3.1 of Chapter 3. It is built on top of an arbitrary *base logic* which has to satisfy some minimal requirements laid down in Sections 3.1.1 and 3.1.2. One of the things that we require is that the base logic be of higher order. *Lax Logic*, defined in Section 3.1.3, is an intuitionistic first-order logic with induction scheme for natural numbers and lists, and a one-place modal operator $\Diamond$. *Lax Logic* contains the *base logic* as a sub-logic, and it is shown that it is a consistent and conservative extension, *cf.* Theorems 3.1.16 and 3.1.18. The logic is presented first in sequent and in Section 3.3 also in natural deduction style.

In Section 3.2 we show how to extract constraint information from a proof in *Lax Logic* and to use it to constrain the theorem proven. Constraint extraction, which is defined for both for the sequent and natural deduction presentation of *Lax Logic*, is parameterized in a *notion of constraint* that can be instantiated in different ways to serve different purposes. In any case, constraining translates a formula of *Lax Logic* into a proposition in the *base logic*. We prove correctness of this process, *viz.* that the extracted constraint information is a well-formed term and that it translates a theorem of *Lax Logic* into a theorem of the *base logic*, *cf.* Theorems 3.2.1 and 3.2.2. An important feature of constraint extraction is that it ignores all proofs done in the *base logic*, so that a controlled form of proof irrelevance is available. In Section 3.2.1 we give a summarizing account of how we intend to apply the extraction process in later examples.

Chapter 4 discusses several simple but illuminating examples of applying the

logic and constraint extraction for a particular notion of constraint.

Meta-theoretical investigations of *Lax Logic* are undertaken in Chapter 5. Section 5.1 looks into a Kripke-style semantics for the propositional fragment of *Lax Logic*. Correctness theorem 5.1.9 is a minimal result to provide justification for calling $\Diamond$ a modality of *possibility* in the standard Kripke sense. In a second line of meta-theory for *Lax Logic* we translate the syntactic calculus into a category theoretical structure, a *hyperdoctrine* with some additional properties. More concretely, it is shown how a hyperdoctrine model for *Lax Logic* can be obtained starting from a hyperdoctrine model of the *base logic*. It is verified that the result precisely captures constraint extraction, *cf.* Theorem 5.2.21 and 5.2.23. Also, it is shown that in this model $\Diamond$ becomes a strong monad, *cf.* Theorem 5.2.19.

This thesis finishes with Chapter 6 on related work and in Chapter 7 the main points of this research are summed up, further research is suggested, and open problems indicated.

Chapter 2

Motivation

Let us illustrate the problem of constraints and the purpose of lax logic by means of simple examples of abstraction mechanisms. The first one (Section 2.1) is to do with representing natural numbers by finite bit-vectors, a central data abstraction in hardware design. The second (Section 2.2) is about simulating integers by natural numbers, which can be viewed as a simple temporal abstraction. The last example (Section 2.3) illustrates the particular form of timing abstraction that is fundamental to synchronous circuit design.

All three examples, which are just complex enough to convey the basic idea, are taken up again later to be formally verified in lax logic. In addition, the second example, which is the simplest, will serve as an expository example in the main part of this thesis.

2.1 Example 1

Consider the usual specification of the factorial function that one would like to work with at the abstract level of natural numbers:

$$fac(0) = 1 \ \wedge \ \forall n.\, fac(n+1) = (n+1) \cdot fac(n).$$

Now, if the factorial is to be implemented by a circuit operating over finite bit-vectors then, of course, this specification is too optimistic. What the eventual

implementation actually will satisfy is a specification like

$$fac(0) = 1 \ \wedge \ \forall n.\, n \leq 10 \ \supset \ fac(n+1) = (n+1) \cdot fac(n).$$

It contains an upper bound on the input to ensure that the implementation does not suffer from an arithmetical overflow.

However, this specification in turn is too realistic since it explicitly contains implementation specific details. It defeats the idea of the specification being independent of the implementation. Firstly, when the specification is being set up we cannot know what the constraint will turn out to be, for it is determined by the implementation and the implementation is unknown at specification time. Secondly, even if we knew the constraint beforehand, putting it into the specification is a bad move: it prohibits us from changing the implementation without affecting the higher levels of the design that are based on the specification.

Thinking about it for a moment one might come up with a third version of specifying the factorial:

$$fac(0) = 1 \ \wedge \ \exists N.\, \forall n.\, n \leq N \ \supset \ fac(n+1) = (n+1) \cdot fac(n).$$

It says 'there is an upper bound N on the input but I don't know which'. But this still is not good enough. It is a bad compromise for at least three reasons. Firstly, it will only work for implementations for which the constraint has exactly the form $n \leq N$. Secondly, the constraint is still sitting in the specification so we have to mess around with it whether we want to or not. Thirdly, given a particular implementation and a proof that it satisfies the specification, there is no guarantee that we will be able to extract the upper bound N from the proof.

Lax logic resolves these problems by taking as the specification of the factorial the lax formula

$$SPEC(f) \ \stackrel{df}{\equiv} \ f(0) = 1 \ \wedge \ \forall n.\, \Diamond(f(n+1) = (n+1) \cdot f(n))$$

where the modal operator $\Diamond$ stands for an *a-priori* unspecified constraint that is going to be determined by a proof of $SPEC(fac)$ for a concrete implementation

fac of the factorial. More precisely, from a proof of $SPEC(fac)$ we will be able to extract a constraint predicate $\gamma(n)$ and a proof of

$$fac(0) = 1 \ \wedge \ \forall n.\, \gamma(n) \ \supset \ fac(n+1) = (n+1) \cdot fac(n).$$

Thereby we have achieved the following goals:

- The abstract specification *SPEC* of the factorial may be used in a design without stumbling over constraints.
- The implementation *fac* may be replaced at any time without having to change the specification too.
- We can let the proof of $SPEC(fac)$, for a given concrete implementation *fac*, decide for the constraint. So, more conservative and less ingenious proofs may result in more conservative and more restrictive constraints.
- Constraint manipulation can be understood as computational behaviour of proofs.

2.2 Example 2

Suppose we wanted to implement and verify a decrementing function *dec* for natural numbers obeying the specification

$$\forall n.\, succ(dec\, n) = n$$

where *succ* is the successor function. Of course, such a totally defined decrementing function cannot exist as there is no predecessor for zero. It would be inconsistent with the standard axioms for natural numbers, among which is $\forall n.\, succ\, n \neq 0$. An approximation of the decrementor can be found, though, such that

$$\forall n.\, n \geq 1 \ \supset \ succ(dec\, n) = n.$$

In most applications this is good enough and the fact that there is an exception at $n = 0$ often can be ignored completely. In these cases it is advantageous to free

formal reasoning from having to bother about the constraint $n \geq 1$ by passing to the formula

$$\forall n. \Diamond (succ(dec\, n) = n).$$

Starting from this 'lax specification' means pretending there was a genuine predecessor for every natural number. One could, for instance, then proceed to define addition and subtraction functions via primitive recursion and to verify

$$\forall m, n. \Diamond (add(sub(m, n), n) = m).$$

In other words, one could simulate integer arithmetic on natural numbers using the constraint level of lax logic to keep track of the points of exception. How this is achieved will become clear from this and the other examples treated in Chapter 4.

Simulating integers by naturals may seem a contrived application but in fact there are reasons to believe that it may be quite useful in hardware verification. There, time is modelled typically by natural numbers, the origin 0 representing start-up, reset, or power-on time. However, the effect of the initial state levels out and after a while the behaviour of a circuit is completely determined by its input. From then on, any condition imposed on the output of the circuit translates into a condition on its input during some interval of time earlier. This output-to-input and backward-in-time reasoning is very convenient for circuit verification. But it faces the technical complication of not being able to go back arbitrarily in time which arises from taking the natural numbers as a model of time. It appears that taking the integers for modelling time in many cases is the more natural approach for verifying stationary input-output behaviour. We believe that this abstraction, which in effect is ignoring start-up initialization of circuits, can be made safe by using lax logic. The decrementor function is a generic example of such use.

2.3 Example 3

The original motivation for the work in this thesis stems from the design of synchronous hardware and the particular form of timing abstraction that is characteristic to that application. This section essentially repeats and extends the motivation of our earlier work [Men91a].

We briefly explain the general situation (Section 2.3.1) and then turn to a concrete example (Section 2.3.2) of a synchronous hardware design.

2.3.1 Synchronous Circuits

This section, in a nutshell, sums up the basic principles underlying synchronous hardware design. The fundamental timing abstraction involved, namely from a synchronous circuit to its abstract behaviour in terms of a finite state machine, and the corresponding timing constraints are explained and formalized.

A typical synchronous circuit is built up from *latches* (such as D-type flipflops) and *combinational circuits* (such as NAND gates, inverters, and nets thereof). In a slightly simplified view[1] one can summarize the essence of the synchronous design paradigm in the following *design rules*:

C1 All latches are triggered by a common clock signal.

C2 There is at least one latch in every feedback loop.

C3 The clock period is long enough to allow for signal changes caused by any clock event to settle throughout the circuit before the next clock event.

C4 The inputs to the circuit have to be stable long enough prior to any clock event for signals to have become stable by the clock event.

[1] We ignore here, among other things, for the sake of simplicity set-up and hold times of latches or the possibility to use multiple clocks. This does not, however, affect the point.

In a broad sense all of these design rules can be interpreted as constraints, more precisely, C1—C2 as *structural* constraints and C3—C4 as *behavioural* constraints. From a verification point of view the structural constraints C1—C2 are essentially reflections of internal behavioural constraints, *i.e.* they are conditions necessary for verifying that no behavioural constraints are violated by components within the circuit.

Much of the success of the synchronous design style is due to the fact that under the design rules C1—C4 one does not need to consider propagation delays when reasoning about the circuit's behaviour. If one is interested in the state of the circuit only at every clock event (or during a certain interval around it) and records the evolution of input and output values at these points, then the descriptive effort can be drastically reduced:

A1 Latches behave like unit delays.

A2 Combinational circuits behave like delay-free boolean functions.

A3 The complete synchronous circuit reduces to a finite automaton and the automaton's behaviour can be derived by composing unit delays and delay-free boolean functions. More precisely, every unit delay gives rise to one state variable and the state transition function is determined by the interconnection of state variables through boolean functions.

Thus, relativizing the synchronous circuit's behaviour to the *abstract time* given by the succession of clock events abstracts from propagation delays. Note, the restriction on clock events can also be viewed as part of the design rules and as a constraint on the usage of the circuit which is characteristic to synchronous abstraction.

Although, either implicitly or explicitly, *timing abstraction* has always been used in the design of synchronous systems [Brz76,Fle80,Mar86], it seems that first attempts to formalize it for the purpose of verification have only recently been

made [Mel88,Her88b]. The separation of design rule checking (C1—C4) from reasoning in abstract terms (A1—A3) is crucial for practical applications, but there seems to be no satisfactory implementation of this separation on an interactive theorem prover. For instance, Herbert's methodology [Her88b], implemented in the proof checker HOL [Gor85,Gor88], though it conceptually distinguishes between statements about timing and abstract behaviour, leaves both aspects intertwined at the level of proofs. Basically this means that design rule checking and reasoning in abstract terms have to go together in a single proof. The logic presented in this thesis provides a way to separate these concerns within a single logical inference system.

2.3.2 A Simple Circuit Design

We take the simple case of a combinational circuit such as a xor-gate and a level-triggered latch (Figure 2–1) which, as in [Her88b], are to be considered as components of a synchronous system; *i.e.* they are put into an environment with a global clock, relative to which certain conditions on the stability of inputs can be imposed as timing constraints to allow timing abstraction. We then go on to consider the simple design task of building a stoppable *modulo-2 counter* from these components and explain how the presence of timing constraints poses a methodological problem in the verification of this design example, why the usual approach is unsatisfactory, and how lax logic is designed to cope with the problem.

The behaviour of the xor-gate and the level-triggered latch will be described by predicates over input and output signals both at the concrete level of asynchronous circuits as well as on the abstract level of finite state machines. At the abstract level the xor-gate is the delay-free exclusive-or boolean function and the latch a one-unit delay. It is shown that the concrete level components only approximately satisfy these abstract specifications and that the offset is given by canonical timing constraints.

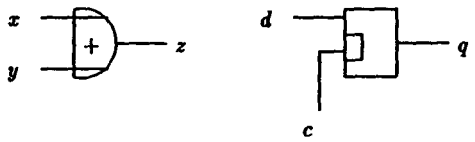


Figure 2–1: Xor-Gate and Level-Triggered Latch

For simplicity we take signals to be functions from integers to booleans, *i.e.*

$$signal \equiv int \Rightarrow bool$$

where $\Rightarrow$ is the constructor for function types. Assuming that both gates have constant propagation delays $\delta_X > 0$ and $\delta_L > 0$, their behaviour may be defined by the following axioms:

$$\begin{aligned} xor\,(x,y,z) &\stackrel{df}{\equiv} \forall t.\ z(t+\delta_X) = x\,t + y\,t \\ latch\,(d,c,q) &\stackrel{df}{\equiv} \forall t.\ (c\,t = 1 \supset q(t+\delta_L) = d\,t)\ \wedge \\ &\qquad (c\,t = 0 \supset q(t+\delta_L) = q(t+\delta_L - 1)) \end{aligned}$$

Note, that we are using the operator $+$ both for addition over *int* as well as for modulo-2 sum over *bool*. According to the axioms the xor-gate performs the modulo-2 sum of its inputs x, y at every time step and outputs them with a delay δ_X on output z. The latch is enabled to pass data from input d to output q with a delay δ_L by positive levels of the clock input c, and it is locked when $c\,t = 0$.

For the purpose of this discussion these simple axioms are assumed to be the low-level, most detailed, description available for the components *xor* and *latch*. Clearly, they are already an abstraction of the real devices' behaviours. A more realistic description would have to account for variable gate delays as well as setup and hold times for the latch; it would perhaps assume continuous rather than discrete time and signal values, require maximal signal rise and fall times, and so on. Since for our logic it is of no importance how detailed a model of behaviour one actually starts from, we have taken the simplest axioms possible. The reader

is referred to [BJ83,HD86] for a discussion of more sophisticated axiomatizations of elementary digital circuits.

The important thing to note is that *xor* and *latch* contain both timing (delays) and functional aspects (operations on booleans) intertwined. In a synchronous design context, however, where one takes advantage of the design rules, one expects not having to care about delays. More precisely, the xor-gate should behave like a delay-free boolean function and the latch like an one-unit delay, depicted in Figure 2-2.

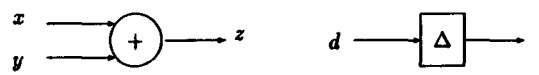


Figure 2-2: Exclusive-Or and One-Unit Delay

In place of *xor* and *latch* one would rather work with axioms like

$$xor\_syn\,(x,y,z) \stackrel{df}{\equiv} \forall t.\, z\,t = x\,t + y\,t \tag{2.1}$$

$$latch\_syn\,(d,q) \stackrel{df}{\equiv} \forall t_1,t_2.\, next\,(t_1\,,\,t_2) \supset q(t_2) = d(t_1) \tag{2.2}$$

where $next\,(t_1\,,\,t_2)$ is a predicate expressing that t_1 and t_2 are two consecutive points in time. It is defined abstractly[2] by

$$next\,(t_1\,,\,t_2) \stackrel{df}{\equiv} t_1 < t_2 \wedge \forall t.\, t_1 < t \supset t_2 \leq t.$$

In this abstract view the clock no longer appears as an input to the latch. For in a synchronous circuit the latch's clock input is always connected to the global clock signal and consequently no longer available as an input. Thus, assume a clock signal

$$clk : signal$$

[2]One might want to turn the predicate *next* into a function which for time t yields the successive abstract time step $next(t)$ if it exists and is undefined otherwise. In a logic with partial terms this could be done using the so-called ι-operator which we do not have available here.

which is globally defined throughout the system. As a result *clk* may be used within formulae without it being mentioned explicitly as a parameter.

Obviously, *xor_syn* and *latch_syn* cannot be proven from *xor* and *latch* right away since the delays cannot be wiped out. What can be proven, however, by introducing constraints are certain approximations thereof. Before we can state them we need some predicates for formulating constraints. We first assume that clock ticks are marked by rising edges of *clk* and define a corresponding predicate

$$tick\, t \stackrel{df}{\equiv} clk(t-1) = 0 \wedge clk\, t = 1$$

which obtains true if there is a clock tick at time t. Given this predicate one may define what it means that a signal x is stable in all intervals of length δ prior to clock events:

$$stable\, x\, \delta \stackrel{df}{\equiv} \forall t_1, t_2.\, (tick\, t_1 \wedge t_1 - \delta \leq t_2 \leq t_1) \supset x(t_1) = x(t_2).$$

Finally, for constraining the clock we have two other predicates, the first expressing that the 1-phase of the clock lasts exactly one time step, and the second imposing a minimal distance δ on two consecutive clock ticks:

$$one\_shot \stackrel{df}{\equiv} \forall t.\, clk\, t = 1 \supset (clk(t-1) = 0 \wedge clk(t+1) = 0)$$
$$min\_sep\, \delta \stackrel{df}{\equiv} \forall t_1, t_2.\, (t_1 < t_2 \wedge tick\, t_1 \wedge tick\, t_2) \supset t_2 \geq t_1 + \delta.$$

With these predicates put into place the promised approximations of *xor_syn* and *latch_syn* can be formulated:

$$\begin{aligned} xor\_abs\,(x,y,z) \stackrel{df}{\equiv}\ & (\boldsymbol{stable\, x\, \delta_X} \wedge \boldsymbol{stable\, y\, \delta_X}) \\ & \supset \forall t.\, \boldsymbol{tick\, t} \supset z\, t = x\, t + y\, t \\ latch\_abs\,(d,q) \stackrel{df}{\equiv}\ & (\boldsymbol{one\_shot} \wedge \boldsymbol{min\_sep\, \delta_L}) \\ & \supset \forall t_1, t_2.(\boldsymbol{tick\, t_1} \wedge \boldsymbol{tick\, t_2} \\ & \quad \supset (next\_abs\,(t_1, t_2) \supset q(t_2) = d(t_1))) \end{aligned}$$

where *next_abs* is the following approximation of *next*:

$$next\_abs\,(t_1, t_2) \stackrel{df}{\equiv} t_1 < t_2 \wedge \forall t.\, \boldsymbol{tick\, t} \supset (t_1 < t \supset t_2 \leq t)$$

The bold-faced parts indicate the offset of the approximations from the ideal versions *xor_syn* and *latch_syn*. This offset explicitly reflects the design rules C3—C4: timing constraints on inputs, on the clock signal, and the sampling at clock events. In contrast to *xor_syn*, *latch_syn* these approximations now can be derived from axioms *xor* and *latch*, *i.e.* we have

$$\begin{aligned} \mathit{xor}\,(x,y,z) &\vdash \mathit{xor\_abs}\,(x,y,z) \\ \mathit{latch}\,(d,\mathit{clk},q) &\vdash \mathit{latch\_abs}\,(d,q). \end{aligned}$$

This follows by straightforward first-order logic. Note, that due to the simplification of the latch's behaviour (*i.e.* no set-up and hold conditions) *latch_abs* does not require stability of data input *d* relative to the clock.

The observation that stability constraints essentially work to squeeze out delays of the behavioural description and thereby separate timing behaviour from functional behaviour is already employed in [Her88b,Mel88]. As seen this idea can be pushed further so as to encompass also constraints on time points, *i.e. tick t* in this case. Being restrictions which also reflect the design rules, constraints on time points should be subjected to the same treatment as are stability constraints on signals. In fact, our logic will also deal with this type of constraints.

Now suppose as a simple design task, we wanted to build a stoppable *modulo-2* counter from *xor* and *latch*. It is to have one input and one output, and to produce a stream of alternating 0s and 1s as long as the input is at 1 and stop at the current output value when the input switches to 0. More formally, its behaviour is specified by the following formula:

$$\mathit{cnt\_spec}\,(x,y) \stackrel{df}{\equiv} \forall t_1,t_2.\ \mathit{next\_abs}\,(t_1,t_2) \supset y(t_2) = x(t_2) + y(t_1).$$

From this input-output specification one derives easily a Moore automaton or equivalently an implementation consisting of an exclusive-or function and a one-unit delay as depicted in Figure 2–3. Given, that *xor_syn* (2.1) describes the behaviour of the exclusive-or and *latch_syn* (2.2) that of the one-unit delay, the

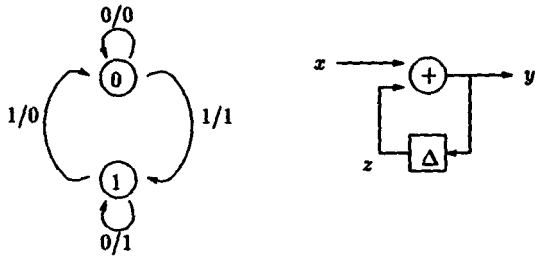


Figure 2–3: Implementation of the *Modulo*-2 Counter

behaviour of the implementation is given by

$$cnt\_syn\,(x,y,z) \stackrel{df}{\equiv} xor\_syn\,(x,z,y) \wedge latch\_syn\,(y,z) \tag{2.3}$$

which employs logical conjunction $\wedge$ of predicates to express *composition* or *superposition* of two behaviours. Verifying that the implementation is correct now would amount to proving that the implementation (2.3) entails the specification, *i.e.*

$$cnt\_syn\,(x,y,z) \vdash cnt\_spec\,(x,y) \tag{2.4}$$

This is an easy exercise invoking the rules of ordinary first-order logic. Unfortunately, applying synchronous abstraction to *xor* and *latch* does not provide an ideal exclusive-or or an ideal one-unit delay satisfying *xor_syn* and *latch_syn* but merely approximations *xor_abs* and *latch_abs*. Therefore the implementation we are actually able to get is

$$cnt\_abs\,(x,y,z) \stackrel{df}{\equiv} xor\_abs\,(x,z,y) \wedge latch\_abs\,(y,z).$$

Of course there is no reason to expect that $cnt\_abs\,(x,y,z)$ entails $cnt\_spec\,(x,y)$. Rather, in place of the original *cnt_spec*, we will again only achieve an approximation, perhaps something of the form

$$\begin{aligned} cnt\_appr\,(x,y) \stackrel{df}{\equiv}\ & (C_0 \wedge C_1(x,y)) \supset (\forall t_1,t_2.\ C_2(t_1,t_2) \\ & \supset (next\_abs\,(t_1,t_2) \supset y(t_2) = x(t_2) + y(t_1))) \end{aligned}$$

where C_0, C_1, and C_2 are constraints that have to be imposed on the composite circuit to allow the envisaged derivation

$$cnt\_abs\,(x, y, z) \vdash cnt\_appr\,(x, y). \tag{2.5}$$

Here we are facing the question of how to go about finding the constraints C_0, C_1, C_2 and thus the modified specification *cnt_appr*. The straightforward approach, as employed for instance in [HD86], is attempting a derivation of *cnt_spec* (from *cnt_abs*), finding out where it fails, and at each such dead end identifying assumptions that would make it work if they were available in the first place. This information can then be used for determining the constraints C_0, C_1, C_2 and the place where they have to go to weaken the specification appropriately. Now, this procedure is not quite satisfactory since it means going through the verification proof twice, once for finding the constraints and a second time after pasting them into the specification for completing the proof. Furthermore, and more importantly, the proof has to intermingle timing constraints with abstracted properties; it aims to verify the abstract specification *cnt_spec* while at the same time it has to deal with the constraints inside the propositions *xor_abs*, *latch_abs*, *next_abs*, and *cnt_appr*.

As argued before, this is not what one really would like to do. Rather, one would like first to perform the abstract verification of (2.4) *without* considering constraints. This establishes the feasibility of the design at the abstract level. The constraints, which are dependent on a particular implementation mechanism, here the implementation as a synchronous circuit, are not determined before the implementations of the abstract components are chosen. In the example, this leads to the approximations *xor_abs*, *latch_abs*. Finally, a constraint analysis should be able to use the abstract proof of (2.4) together with the knowledge of the constraints contained in *xor_abs* and *latch_abs* for extracting the constraints in *cnt_appr*.

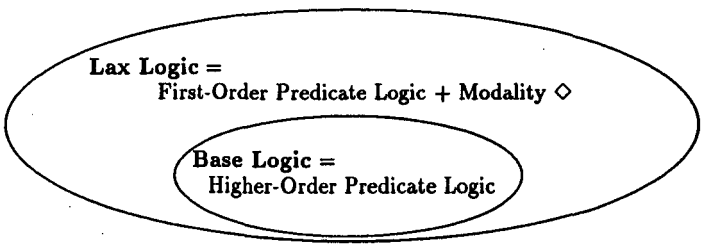
In this thesis it will be shown how this goal can be achieved by reformulating the notions of proof and proposition so as to 'hide' constraints within them and

set up a calculus of derivations to deal with this *lax* logic.

Chapter 3

Lax Logic

Lax logic is a first-order extension of a version of higher-order predicate logic, called the base logic. The heart of the extension is a modal operator $\Diamond$ that relaxes the meaning of a proposition so as to account for potential constraints and constraint analysis.



The base logic is arbitrary to some extent, so that lax logic need not be seen as a particular fixed calculus but rather as a method of "laxifying" ones favourite predicate logic. In contrast to the base logic, lax logic cannot be arbitrary since there we wish to extract constraint information. As a consequence, lax logic is an *intuitionistic* extension, while the base logic may well have an axiom of choice or the axiom of the excluded middle. Another consequence is that we will have to record proof objects in lax logic while the semantics of the base logic is chosen to be proof irrelevant. So, from an implementation viewpoint it is desirable to single out reasoning within the base logic. This is the main reason why we are

going to distinguish clearly between the base logic and lax logic. The idea is that the base logic is the intended realm of discourse in which formal verification takes place whenever possible and that its first-order extension of lax logic is entered only when there is need to consider approximate specifications and to handle constraints.

Before we start off with formal definitions we briefly explain the central issues of lax logic. Let ϕ and ψ stand for arbitrary propositions of the base logic in what follows. Given specification ϕ, the abstract proof-theoretic intuition behind $\bigcirc\phi$ is "for some constraint c, ϕ is provable under c". In other words, a proof of $\bigcirc\phi$ is a pair (c, p) where c is a constraint and p is a proof of ϕ under constraint c. At this level of generality, nothing is said about what kind of object a constraint actually is and what it means that a specification is provable "under" a constraint. Different notions of constraint will have different properties, and thus will give rise to different derivation rules for $\bigcirc$. The particular interpretation that we are going to focus on in this thesis is discussed below in Section 3.2 and further in Chapter 5. More concrete intuitions may apply to specific interpretations of lax logic, as we have seen in the context of hardware verification.

There are three general rules for $\bigcirc$, which we consider — without interpretative bias — as being characteristic for any notion of constraint, pronouncing $\bigcirc\phi$ "somehow ϕ". The three rules, or rather rule schemes, are

$$\frac{}{\phi \vdash \bigcirc\phi}\bigcirc I \qquad \frac{}{\bigcirc\bigcirc\phi \vdash \bigcirc\phi}\bigcirc M \qquad \frac{\Gamma, \bigcirc\phi \vdash \bigcirc\psi}{\Gamma, \phi \vdash \psi}\bigcirc F.$$

The reader is warned that throughout this thesis rules are written just the other way round as usual, *i.e.* they extend derivations for the sequents below the rule bar into a derivation of the sequent above the rule bar. We have chosen to turn derivation trees upside down since this presentation is more natural for refinement proofs, which will be our main concern in this thesis.

Formally, $\bigcirc I$ introduces a $\bigcirc$ operator, $\bigcirc M$ collapses two occurrences of $\bigcirc$ into one, and $\bigcirc F$ lifts a derivation of conclusion ψ from hypothesis ϕ by prefixing

both the hypothesis and the conclusion with $\Diamond$. The following three very natural assumptions on the structure of constraints are sufficient to justify these rules:

- There is a *void* constraint 1 such that 'ϕ *under* 1' is equivalent to ϕ.
- There is a *multiplication* of constraints such that '(ϕ *under* c) *under* d' is equivalent to 'ϕ *under* $c \cdot d$'.
- Constraining *preserves implication*, *i.e.* if ϕ implies ψ, then 'ϕ *under* c' implies 'ψ *under* c'.

Recall that a proof of $\Diamond\phi$ is to be a pair (c, p) with c a constraint and p a proof of 'ϕ *under* c'. By the first condition we know that whenever ϕ is provable, then ϕ is provable under the vacuous constraint 1, whence rule $\Diamond I$. The fact that 'ϕ *under* 1' actually is equivalent to ϕ means that by passing from ϕ to $\Diamond\phi$ we have not lost any information, provided we record the constraint. Similarly, the second condition can be seen to justify rule $\Diamond M$, allowing us to melt together two nested occurrences of $\Diamond$ without loosing information. Finally, rule $\Diamond F$ basically says that if from the hypothesis ϕ we can prove ψ, then from the (weaker) hypothesis that ϕ is provable under some constraint c we can prove ψ under c. This is guaranteed by the third condition. We might sum up the three conditions as follows:

> **Postulate:** A *notion of constraint* is a monoidal action $(\mathbf{C}, 1, \cdot, - \textit{ under } -)$ on formulae or truth values (of the base logic) that preserves implication.

For the sake of definiteness let us list a few concrete notions of constraint that one might consider:

$\mathbf{C}$	$\{\top, \bot\}$
1	$\top$
$\cdot$	$\bot \cdot \top = \bot \cdot \bot = \top \cdot \bot = \bot$ and $\top \cdot \top = \top$
$- \textit{ under } -$	$\phi \textit{ under } \bot \equiv \textit{true}$
	$\phi \textit{ under } \top \equiv \phi$

Here and in the rest of this thesis the symbol $\equiv$ is used to denote syntactic, or definitional identity. This is perhaps the most simple non-trivial notion of constraint; it takes as constraints a two-valued set $\{\bot, \top\}$ behaving like the booleans, it has $\top$ as the void constraint that does not change the meaning of a proposition while the only other constraint $\bot$ evaporates a proposition to become trivially provable.

Taking as a constraint the set-up time of a synchronous circuit results in a very specific notion of constraint, *viz.*

C	N
1	0
$\cdot$	$n \cdot m = max\ \{n, m\}$
$-\ under\ -$	$\phi\ under\ n \equiv setup\ n \supset \phi$

where N denotes the natural numbers, and '*setup n*' is a predicate expressing that all inputs to a given circuit are stable immediately before each clock tick during an interval the length of which is at least n time units.

In this thesis we are going to focus on the following very general notion of constraint, which will be discussed and used at length later on:

C	lists $[\gamma_1, \ldots, \gamma_n]$, where γ_i are from some subclass of propositions
1	[] (empty list)
$\cdot$	@ (concatenation)
$-\ under\ -$	$\phi\ under\ [\gamma_1, \ldots, \gamma_n] \equiv \gamma_n \supset \cdots \supset \gamma_1 \supset \phi$

A notion of constraint $(\mathbf{C}, 1, \cdot, -\ under\ -)$ defines what it means that a proposition ϕ of the base logic holds under a constraint $c \in \mathbf{C}$, *i.e.* the meaning of '$\phi\ under\ c$'. In lax logic we will be concerned with lifting this notion of constraining to arbitrary propositions, rather than just propositions of the base logic. An example, which we have seen before, is the specification of the decrementor:

$$\forall n. \Diamond (succ(dec\, n) = n).$$

From a proof of this proposition, call it p, we wish to obtain a function f that associates with each input n a constraint $f\,n \in \mathbf{C}$ such that

$$\forall n.\,(succ(dec\,n) = n)\ \textit{under}\ f\,n.$$

For the first notion of constraint mentioned above, *viz.* the one that has $\mathbf{C} = \{\top, \bot\}$, f might be the characteristic function of the subset $\{n \mid n \geq 1\}$ of inputs that are decremented correctly. The function term f will be called the *constraint term*, written $f = |p|$, and the predicate

$$(\forall n.\,\Diamond\,(succ(dec\,n) = n))^*[z] \ \equiv\ \forall n.\,(succ(dec\,n) = n)\ \textit{under}\ z\,n$$

will be called the *constraint predicate*. It can be thought of as the lifting of $-\ \textit{under}\ -$ to proposition $\forall n.\,\Diamond\,(succ(dec\,n) = n)$.

To give another example let us assume that we have built the factorial function using the decrementor as a subcomponent, *i.e.* that we have a proof of

$$\forall n.\,\Diamond\,(succ(dec\,n) = n) \ \supset\ \forall n.\,\Diamond\,(fac(n+1) = (n+1)\cdot fac(n))$$

where, for simplicity, we have dropped the base case in the specification of the factorial. From this proof we expect to extract a higher-order functional F that translates every input constraint f, *i.e.* the characteristic function of a subset of inputs, of the decrementor into an input constraint of the factorial, $F\,f$, which is again the characteristic function of a subset of inputs. This can be expressed by requiring that the proposition

$$\begin{aligned}&\forall f.\ \forall n.\,(succ(dec\,n) = n)\ \textit{under}\ f\,n\\ &\qquad\supset\ \forall n.\,(fac(n+1) = (n+1)\cdot fac(n))\ \textit{under}\ F\,f\,n\end{aligned}$$

hold in the base logic. Thus, for more complex propositions the extracted constraint term and the constraint predicate can be quite complex objects. For arbitrary propositions ϕ, the constraint predicate $\phi^*[z]$ will be defined by induction on the structure of ϕ, and the constraint term $|p|$ is given by induction on the structure of proof p.

So much for an informal introduction to the rationale behind lax logic. The rest of this chapter deals with the formal systems of base logic and lax logic which are set up formally in Section 3.1. The syntactic calculus of lax logic is presented both in sequent and, for use in later examples, in natural deduction style (*cf.* Section 3.3). In Section 3.2 the general notion of constraint, which was briefly mentioned above, is discussed. Also, the associated constraint extraction and analysis process for lax logic is defined, based on the constructive nature of derivations.

3.1 Definition of Lax Logic

Our main concern lies in lax logic as an extension on top of a base logic, which satisfies certain minimal requirements but is otherwise arbitrary. In order to stress this aspect it would be natural to define the calculus of lax logic first, treating the base logic as a parameter. Nevertheless, we start off with the base logic as this has the advantage of requiring less notational overhead and of introducing the syntax in its logical ordering.

The base logic will be introduced in two steps, *viz.* first a rudimentary part in Section 3.1.1 amounting to a fragment of second-order logic, and then an enrichment by a full-fledged lambda-calculus giving higher-order expressibility in Section 3.1.2. The extension of lax logic, finally, will be treated in Section 3.1.3.

3.1.1 Base Logic

SIGNATURE. The language of the base logic is specified by a *signature* Σ, which is a collection of *sorts* A, B, *etc.* and a collection of *operators* f, g, *etc.* together with a map assigning to each operator f a finite non-empty sequence $A_1 \cdots A_n$ of sorts, called its *arity*. If $n = 1$, then f also is called a *constant*. It is assumed that among the sorts there is a distinguished sort Ω, the sort of propositions. What other sorts and operators there are in the signature depends on the particular

base logic chosen.

TYPES. Every sort A is a well-formed (primitive) *type*. For now the sorts will be the only types. Later the base logic will be extended by type-forming operations for building non-primitive types. Meta-variables σ, τ, *etc.* will be used to range over types.

TERMS. Given a signature Σ we define the set of terms over Σ, ranged over by meta-variables $s, t, \ldots, \phi, \psi$, *etc.* First, fix for each type τ a countably infinite number of *variables*, denoted by x^τ, y^τ, *etc.* Assuming that a variable uniquely determines the type to which it belongs, the superscript τ may be omitted. Terms now are built from variables, operators, and the two special term forming symbols *universal quantifier* $\forall$ and *implication* $\supset$ in the usual way. More precisely, one defines the class of *well-formed terms* t of type τ along with the *free* variables of a term as follows:

- Any variable x^τ is a well-formed term of type τ; its only free variable is x.
- If f is an operator with arity $\tau_1 \cdots \tau_n \tau$ and $t_1, \ldots, t_n$ well-formed terms of type $\tau_1, \ldots, \tau_n$, respectively, then $f(t_1, \ldots, t_n)$ is a well-formed term of type τ; its free variables are those of t_i, $i = 1, \ldots, n$.
- If ϕ is a well-formed term of type Ω and x a variable, then $\forall x.\, \phi$ is a well-formed term of type Ω; its free variables are the free variables of ϕ except x.
- If ϕ and ψ are well-formed terms of type Ω, then $\phi \supset \psi$ is well-formed of type Ω; its free variables are those of ϕ and ψ.
- If s and t are well-formed terms of the same type, then $s = t$ is well-formed of type Ω; its free variables are those of s and t.

The type τ of a well-formed term t and its set of free variables, denoted by $FV(t)$, are thus uniquely determined[1]. We note that if x is a free variable of t, then x actually occurs as a sub-term of t. A term without free variables is called *closed*. As usual every occurrence of a variable x within term $\forall x.\,\phi$ is called *bound*. Two terms are called *α-convertible* if they are syntactically identical modulo renaming of bound variables.

For now the terms defined will be the only terms of the base logic. Later the base logic will be extended by other term-forming operations.

We may turn the definition of well-formedness into a formal calculus given by Figure 3–1. The formal judgement $\Delta \vdash t : \tau$ is called a *typing*, where $\Delta = x_1, \ldots, x_n$ is a finite, possibly empty, non-duplicating list of variables called a *context*. Such a typing is to stand for the statement that t is well-formed of type τ with free variables in Δ. We will say that a term is well-formed of type τ *in context* Δ *iff* $\Delta \vdash t : \tau$ can be derived.

As mentioned before, all rules in this thesis are to be read bottom up, *i.e.* a rule instance of Figure 3–1 may be applied to turn derivations of the typings below the rule bar into a derivation of the typing above it. For all forms of rules to be introduced in this thesis we shall call the judgements below the rule bar the *premisses* and the judgement above *conclusion* of the rule.

Observation 3.1.1 *A term t is well-formed of type τ with all its free variables in the context Δ iff $\Delta \vdash t : \tau$ can be derived by the rules of Table 3–1.*

Proof: The simple proof proceeds by induction on the structure of t and the derivation $\Delta \vdash t : \tau$. The only tricky case is to show that $\Delta \vdash \forall x.\,\phi : \Omega$ is derivable if $\forall x.\,\phi$ is well-formed with free variables in Δ. Two case must be distinguished: If $x \notin \Delta$, then use the induction hypothesis on ϕ with free variables in Δ, x. If $x \in \Delta$, then use induction hypothesis on ϕ with free variables in $\Delta \setminus x$.

■

[1]The definition of well-formed terms defines abstract syntax. We will use auxiliary bracketing '(', ')' to indicate the structure of a term.

$$\frac{\Delta \vdash f(t_1,\ldots,t_n):\tau}{\Delta \vdash t_1:\tau_1 \quad \cdots \quad \Delta \vdash t_n:\tau_n}(f \textit{ operator with arity } \tau_1\cdots\tau_n\tau)$$

$$\frac{\Delta \vdash \forall x.\phi:\Omega}{\Delta, x \vdash \phi:\Omega} \qquad \frac{\Delta \vdash \phi \supset \psi:\Omega}{\Delta \vdash \phi:\Omega \quad \Delta \vdash \psi:\Omega}$$

$$\frac{}{x^\tau \vdash x:\tau} x^\tau \textit{ variable} \qquad \frac{\Delta \vdash s=t:\Omega}{\Delta \vdash s:\tau \quad \Delta \vdash t:\tau}\tau \textit{ a type}$$

$$\frac{\Delta, x \vdash t:\tau}{\Delta \vdash t:\tau} x \textit{ variable, not in } \Delta \qquad \frac{\Delta, y, x, \Delta' \vdash t:\tau}{\Delta, x, y, \Delta' \vdash t:\tau}$$

Figure 3–1: Well-Formed Terms of the Base Logic, I

We list a few immediate consequences of this observation. For each well-formed term t there is a unique τ and a unique minimal Δ such that $\Delta \vdash t:\tau$. This minimal Δ is given by the set of free variables $FV(t)$ of t. The closed terms of type τ are those t for which $\vdash t:\tau$ is derivable. If $\Delta, x \vdash t:\tau$ and x not free in t, then $\Delta \vdash t:\tau$. Finally, the substitution rule

$$\frac{\Delta \vdash s\{t/x\}:\sigma}{\Delta, x^\tau \vdash s:\sigma \quad \Delta \vdash t:\tau}$$

is valid, where as usual $s\{t/x\}$ denotes the result of substituting in s all free occurrences of variable x by t. As usual it must be assumed that substitution renames bounds variable occurrences in s to avoid name capture of free variables in t.

PROPOSITIONS. Every well-formed term of type Ω is a *proposition* of the base logic. Propositions will be ranged over by Greek meta-variables ϕ, ψ, *etc.*

SEQUENTS. The logical calculus of the base logic rests on the formal judgement of logical entailment

$$\Gamma \vdash_\Delta \phi$$

where Δ is a context and $\Gamma = \phi_1, \cdots, \phi_n$ a possibly empty list of propositions, *i.e.* well-formed terms of type Ω, with free variables in Δ, *i.e.* we have $\Delta \vdash$

$\phi_i : \Omega$, $i = 1, \ldots, n$. The judgement will be called *sequent*, the propositions in Γ its *hypotheses* and ϕ its *assertion*. The calculus is to be closed under the usual structural rules of identity, weakening, cut, permutation of hypotheses and substitution, plus the logical rules for implication and universal quantification. The rules are shown in Figure 3-2. In rule *sub* the notation $\Gamma\{t/x\}$ means substitution is performed on all hypotheses in Γ. Note, we require the base logic merely to be closed under these rules, so it may have more rules. In particular we do not assume that the base logic is intuitionistic.

$$\frac{\phi \vdash_\Delta \phi}{\Delta \vdash \phi : \Omega}\, id \qquad \frac{\Gamma, \phi \vdash_\Delta \psi}{\Gamma \vdash_\Delta \psi \quad \Delta \vdash \phi : \Omega}\, weak \qquad \frac{\Gamma \vdash_{\Delta, x} \phi}{\Gamma \vdash_\Delta \phi}\, weak \ (x \text{ not in } \Delta)$$

$$\frac{\Gamma \vdash_{\Delta, y, x, \Delta'} \phi}{\Gamma \vdash_{\Delta, x, y, \Delta'} \phi}\, perm \qquad \frac{\Gamma, \psi, \phi, \Gamma' \vdash_\Delta \theta}{\Gamma, \phi, \psi, \Gamma' \vdash_\Delta \theta}\, perm$$

$$\frac{\Gamma\{t/x\} \vdash_\Delta \psi\{t/x\}}{\Gamma \vdash_{\Delta, x^\tau} \psi \quad \Delta \vdash t : \tau}\, sub \qquad \frac{\Gamma \vdash_\Delta \psi}{\Gamma \vdash_\Delta \phi \quad \Gamma, \phi \vdash_\Delta \psi}\, cut$$

$$\frac{\vdash_x x = x}{}\, x \text{ variable} \qquad \frac{\vdash_\Delta \forall x.\, \phi = \forall x.\, \psi}{\vdash_{\Delta, x} \phi = \psi}\, \xi$$

$$\frac{\Gamma \vdash_\Delta \phi\{s/x\}}{\Delta, x^\tau \vdash \phi : \Omega \quad \Gamma \vdash_\Delta s =_\tau t \quad \Gamma \vdash_\Delta \phi\{t/x\}}\, Subst$$

$$\frac{\Gamma \vdash_\Delta \phi \supset \psi}{\Gamma, \phi \vdash_\Delta \psi} \supset I \qquad \frac{\Gamma, \phi \vdash_\Delta \psi}{\Gamma \vdash_\Delta \phi \supset \psi} \supset E$$

$$\frac{\Gamma \vdash_\Delta \forall x.\, \phi}{\Gamma \vdash_{\Delta, x} \phi} \forall I \ (x \text{ not free in } \Gamma) \qquad \frac{\forall x.\, \phi \vdash_\Delta \phi\{t/x\}}{\Delta \vdash t : \tau \quad \Delta, x^\tau \vdash \phi : \Omega} \forall E$$

Figure 3-2: Sequent Rules of the Base Logic

Observation 3.1.2 *Every sequent $\phi_1, \ldots, \phi_n \vdash_\Delta \psi$ derived by means of the rules of Figure 3-2 is well-formed. More precisely, Δ is a context (non-duplicating list of variables), and the assertion ψ as well as all hypotheses $\phi_1, \ldots, \phi_n$ are well-formed propositions in context Δ.*

The observation is due to our including of typings as additional premisses in some of the rules. In usual presentations of typed predicate logic the typing

judgement is left as an implicit side condition rather than being formalized as part of the calculus. For instance, the global side condition may be imposed that a rule can only be applied in case all premisses and the conclusion of the rule are well-formed. In our system we can safely add

$$\frac{\Delta \vdash \phi : \Omega}{\Gamma \vdash_\Delta \phi}$$

to the rules in Figure 3-2 without changing the set of derivably well-formed propositions. Henceforth, it will be assumed tacitly that this rule is available.

The fact that the type of all propositions Ω is a type of the language means we can quantify over propositions. Thus, the other logical connectives *false*, *true*, $\neg$, $\wedge$, $\vee$, $\exists$, $\cong$ of (intuitionistic) predicate logic can be defined in the standard way by $\supset$ and $\forall$, see *e.g.* [Pra65]

$$\begin{aligned}
\mathit{false} &\stackrel{df}{\equiv} \forall z^{\Omega}.\, z \\
\mathit{true} &\stackrel{df}{\equiv} \forall z^{\Omega}.\, z \supset z \\
\neg\phi &\stackrel{df}{\equiv} \phi \supset \mathit{false} \\
\phi \wedge \psi &\stackrel{df}{\equiv} \forall z^{\Omega}.\, (\phi \supset (\psi \supset z)) \supset z \\
\phi \vee \psi &\stackrel{df}{\equiv} \forall z^{\Omega}.\, (\phi \supset z) \supset ((\psi \supset z) \supset z) \\
\exists x.\, \phi &\stackrel{df}{\equiv} \forall z^{\Omega}.\, (\forall x.\, (\phi \supset z)) \supset z \\
\phi \cong \psi &\stackrel{df}{\equiv} (\phi \supset \psi) \wedge (\psi \supset \phi)
\end{aligned}$$

where it is understood that variable z does not occur free or bound in ϕ, ψ. The abbreviations are well-formed: *false* and *true* are closed well-formed terms of type Ω; if ϕ is well-formed of type Ω, then so is $\neg\phi$, its free variables are those of ϕ; if ϕ and ψ are well-formed terms of type Ω, then so are $\phi \wedge \psi$ and $\phi \vee \psi$, their free variables are those of ϕ and ψ; if ϕ is well-formed of type Ω and x a variable, then so is $\exists x.\, \phi$, its free variables are those of ϕ except x.

Observation 3.1.3 *For the logical connectives as defined above the usual elimination and introduction rules, shown in Figure 3-3, can be derived from rules of Figure 3-2.*

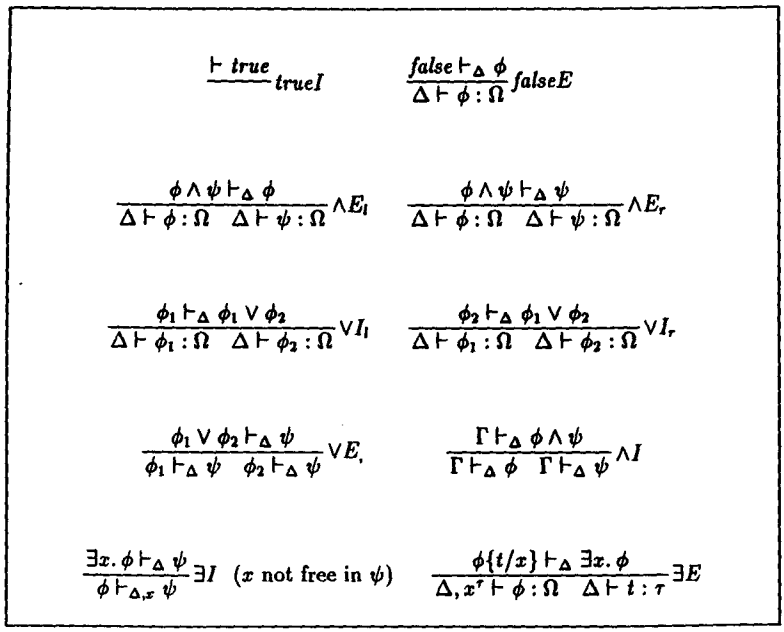


Figure 3–3: Derived Rules for the Base Logic

To sum up, the base logic, henceforth referred to by the symbol $\mathcal{B}$, is specified by a signature Σ of sorts and operators. Its syntactic categories are types, terms a special kind of which are propositions, and sequents. The terms are constructed from variables and operators by syntactic substitution. In particular, propositions are built up from terms of type Ω using logical symbols $\supset$ and $\forall$. It is pointed out again that the definitions of these syntactic categories are to be understood as minimal requirements on $\mathcal{B}$.

Let us briefly comment on the status of $\mathcal{B}$ as specified so far. Since we allow ourselves to quantify over propositions, $\mathcal{B}$ has some kind of second-order expressibility, which was exploited to encode the connectives $\wedge, \vee$, *etc.* On the other hand $\mathcal{B}$ is not full second-order logic. For instance, it is not possible to

define Leibnitz equality internally by

$$x =_L y \stackrel{df}{\equiv} \forall P.\, P(x) \supset P(y)$$

with the understanding that P stands for any propositional context with a distinguished hole. The reason is that there are no predicate variables in $\mathcal{B}$, whence we cannot quantify over propositional contexts. In Prawitz's simple second-order logic [Pra65] P is called a 1-place predicate variable and in Andrews' second order logic $\mathcal{F}^2$ [And86], a variable of type '(ι)'.

In some, albeit less fundamental, sense $\mathcal{B}$ has more structure than Prawitz's and Andrews' systems of second-order logic. Namely, $\mathcal{B}$ can have second-order operators of sort Ω, Ω or Ω, Ω, Ω *etc.* which, according to Andrews' classification, are part of third-order logic but not of second-order. So, for instance, the two-place logical connective $\supset$ could be introduced in $\mathcal{B}$ as a distinguished operator of sort Ω, Ω, Ω rather than a syntactical operation for forming terms.

3.1.2 Extending the Base Logic

The object language of the base logic introduced so far does not comprise any specific data types and associated operators. Here, an enrichment is necessary for two reasons: First, we will need a minimum structure for expressing specific 'real world' verification examples in Chapter 4. Second, as will become clear later, in order to feed back into the base logic the constraint information that we are going to extract from proofs in lax logic, and to reason about it, the object language of the base logic must be rich enough to express the structure of proofs.

In this section $\mathcal{B}$ will be extended by a simply typed lambda calculus with finite products, finite sums, exponentials, natural numbers, and for every type τ the type τ^* of finite lists of elements of type τ. For these data types the usual β and η-equalities, and additionally for lists and natural numbers the standard induction schemes will be assumed. In the following we will spell out these assumptions in detail. Let S, O be the class of sorts and operators, respectively, that constitute the signature Σ of $\mathcal{B}$.

TYPES. The *well-formed types* of $\mathcal{B}$ are generated by S as the primitive types and the type forming operations

$$\tau ::= A \mid \mathbf{0} \mid \tau + \tau \mid \mathbf{1} \mid \tau \times \tau \mid \tau \Rightarrow \tau \mid \tau^* \mid \mathbf{N}.$$

Here A stands for any sort in S. $\mathbf{N}$ is the type of natural numbers and τ^* the type of lists with elements of type τ.

TERMS. The *terms* of $\mathcal{B}$ are built from variables (an infinite number of variables of each type is assumed), the operators in O, the logical symbols $\supset, \forall, =$, and the standard constructors and destructors for the composite types, *i.e.* they are of the shapes

$t ::=$	$x \mid f(t,\ldots,t) \mid$	*variables and operators*
	$t \supset t \mid \forall x.t \mid t = t \mid$	*propositions*
	$\Box t \mid \iota_1 t \mid \iota_2 t \mid case_{x,y}(t,t,t) \mid$	*finite sums*
	$* \mid \pi_1 t \mid \pi_2 t \mid (t,t) \mid$	*finite products*
	$t\,t \mid \lambda x.t \mid$	*exponentials*
	$[] \mid t :: t \mid fold_{x,x}(t,t) \mid$	*lists*
	$0 \mid succ \mid iter_x(t,t)$	*naturals*

where x, y, z are variables and f is an operator in O. The definition in Section 3.1.1 of *well-formed* terms of type τ is extended by the following clauses:

- If t is well-formed of type $\mathbf{0}$ then $\Box t$ is well-formed of any type τ; its free variables are those of t.
- If t is well-formed of type σ and τ a type, then $\iota_1 t$ and $\iota_2 t$ are well-formed of type $\sigma + \tau$ and $\tau + \sigma$, respectively; their free variables are those of t.
- If s and t are well-formed of type τ, x variable of type σ_1, y variable of type σ_2, and u well-formed of type $\sigma_1 + \sigma_2$ such that x, y are not free in u, x not free in t and y not free in s, then $case_{x,y}(u,s,t)$ is well-formed of type τ; its free variables are those of s, t, and u except x and y.

- $*$ is a well-formed closed term of type $\mathbf{1}$.

- If t is well-formed of type $\sigma \times \tau$ then $\pi_1 t$ and $\pi_2 t$ is well-formed of type σ and τ, respectively; their free variables are those of t.

- If s is well-formed of type σ, t well-formed of type τ, then (s,t) is well-formed of type $\sigma \times \tau$; its free variables are those of s and t.

- If s is well-formed of type $\sigma \Rightarrow \tau$ and t well-formed of type τ, then st is well-formed of type τ; the free variables of st are the free variables of s and t.

- If t is well-formed of type τ, and x a variable of type σ, then $\lambda x.t$ is well-formed of type $\sigma \Rightarrow \tau$; its free variables are those of t except x.

- $[\,]$ is a well-formed closed term of type τ^* for any type τ.

- If s is well-formed of type τ and t well-formed of type τ^*, then $s :: t$ is well-formed of type τ^*; its free variables are those of s and t.

- If s and t are well-formed terms of type σ, x variable of type σ and z variable of type τ, both not free in s, then the term $fold_{z,x}(s,t)$ is well-formed of type $\tau^* \Rightarrow \sigma$; its free variables are those of s and t except x and z.

- 0, *succ* are well-formed closed terms of type $\mathbf{N}, \mathbf{N} \Rightarrow \mathbf{N}$, respectively. If a and f are well-formed terms of type τ, x variable of type τ, not free in a, then $iter_x(a,f)$ is well-formed of type $\mathbf{N} \Rightarrow \tau$; its free variables are those of a and f except x.

As before, terms of type Ω are called *propositions*. A well-formed term may have more than one type according to the above definition. Types could be made unique, as it is usually done, by annotating terms with types. For instance, one would distinguish type-many instances $[\,]_\tau$ of the empty list and define $[\,]_\tau$ to have

type σ *iff* $\sigma = \tau$. For terms $t_1 :: \cdots :: t_n :: []$ of type τ^* we shall use the more standard[2] notation $[t_n, \ldots, t_1]$.

Note, the forms *case*, λ, *fold*, and *iter* have variable binding effect, and the notion of a *bound* variable has to be extended to terms $case_{x,y}(u,s,t)$, $\lambda x.t$, $fold_{z,x}(s,t)$, and $iter_x(a,f)$ in the apparent way.

The above definition of well-formedness may be cast into a system of formal rules as shown in Figure 3–4 which we add to the typing rules of Figure 3–1.

$$\frac{\Delta \vdash \Box t : \tau}{\Delta \vdash t : \mathbf{0}}\,\tau\ type \qquad \frac{\Delta \vdash \iota_1 t : \sigma + \tau}{\Delta \vdash t : \sigma}\,\tau\ type \qquad \frac{\Delta \vdash \iota_2 t : \tau + \sigma}{\Delta \vdash t : \sigma}\,\tau\ type$$

$$\frac{\Delta \vdash case_{x,y}(u,s,t) : \tau}{\Delta \vdash u : \sigma_1 + \sigma_2 \quad \Delta, x^{\sigma_1} \vdash s : \tau \quad \Delta, y^{\sigma_2} \vdash t : \tau}$$

$$\frac{}{\vdash * : \mathbf{1}} \qquad \frac{\Delta \vdash \pi_1 t : \sigma}{\Delta \vdash t : \sigma \times \tau} \qquad \frac{\Delta \vdash \pi_2 t : \tau}{\Delta \vdash t : \sigma \times \tau} \qquad \frac{\Delta \vdash (s,t) : \sigma \times \tau}{\Delta \vdash s : \sigma \quad \Delta \vdash t : \tau}$$

$$\frac{\Delta \vdash s\,t : \tau}{\Delta \vdash t : \sigma \quad \Delta \vdash s : \sigma \Rightarrow \tau} \qquad \frac{\Delta \vdash \lambda x.t : \sigma \Rightarrow \tau}{\Delta, x^{\sigma} \vdash t : \tau}$$

$$\frac{}{\vdash 0 : \mathbf{N}} \qquad \frac{}{\Delta \vdash succ : \mathbf{N} \Rightarrow \mathbf{N}} \qquad \frac{\Delta \vdash iter_x(s,t) : \mathbf{N} \Rightarrow \tau}{\Delta \vdash s : \tau \quad \Delta, x^{\tau} \vdash t : \tau}$$

$$\frac{}{\vdash [] : \tau^*}\,\tau\ type \qquad \frac{\Delta \vdash s :: t : \tau^*}{\Delta \vdash s : \tau \quad \Delta \vdash t : \tau^*} \qquad \frac{\Delta \vdash fold_{z,x}(s,t) : \tau^* \Rightarrow \sigma}{\Delta \vdash s : \sigma \quad \Delta, z^{\tau}, x^{\sigma} \vdash t : \sigma}$$

Figure 3–4: Well-Formed Terms of $\mathcal{B}$, II

Observation 3.1.4 *A term t is well-formed of type τ with all its free variables in the context Δ iff $\Delta \vdash t : \tau$ can be derived by the rules of Figures 3–1 and 3–4.*

SEQUENTS. Finally, the base logic $\mathcal{B}$ is extended by the usual equality axioms for the data types as summed up in Figure 3–5, the standard induction principle

[2]The reversing of the order in which the elements appear is inessential but technically convenient.

for finite lists

$$\frac{\Gamma \vdash_\Delta \forall z.\, \phi}{\Gamma \vdash_\Delta \phi\{[]/z\} \quad \Gamma, \phi \vdash_{\Delta, x, s} \phi\{x :: z/z\}}\ \mathit{ListInd}$$

where x, z are variables of type τ, τ^*, respectively, x not free in Γ, ϕ, z not free in Γ, and induction for natural numbers

$$\frac{\Gamma \vdash_\Delta \forall z.\, \phi}{\Gamma \vdash_\Delta \phi\{0/z\} \quad \Gamma, \phi \vdash_{\Delta, s} \phi\{succ\, z/z\}}\ \mathit{NatInd}$$

where z is variable of type N not free in Γ.

We have separated the equality rules in Figure 3–5 into three classes, *viz.* β, η, ξ equalities. This classification is taken over from lambda calculus, where one mainly is dealing with function types. As in lambda calculus the β-equations capture the computational meaning of terms, the η-equations amount to extensionality of the data types, and the ξ-rules are structural rules that enable us to substitute equals for equals within the variable-binding operators λ, *case*, *iter*, and *fold*. For the other operators such ξ-rules already are covered by the ordinary substitution rule *Subst*. For instance, the rule

$$\frac{\vdash_\Delta (s, t) = (s', t')}{\vdash_\Delta s = s' \quad \vdash_\Delta t = t'}$$

is a derived rule in $\mathcal{B}$. Extensionality of natural numbers, *i.e.* the uniqueness of iteration, expressed by the scheme

$$f\, 0 = s \ \wedge\ \forall y.\, f(succ\, y) = t\{f\, y/x\} \ \supset\ f = iter_x(s, t)$$

is derivable from extensionality of functions and induction. A similar remark applies to extensionality of lists.

Observation 3.1.5 *Every sequent $\Gamma \vdash_\Delta \phi$ derived by the rules of Figures 3–2 and 3–5 is well-formed, i.e. Δ is a context (non-duplicating list of variables), and the assertion ϕ as well as all hypotheses in Γ are well-formed propositions in context Δ.*

β Equalities

$$\frac{\vdash_{\Delta,x^0} t = \Box x}{\Delta, x^0 \vdash t : \tau} \qquad \frac{\vdash_{\Delta,x} (\lambda x.t)x \;=\; t}{\Delta, x^\sigma \vdash t : \tau}$$

$$\frac{}{\vdash_{x^\sigma, y^\tau} \pi_1(x,y) \;=\; x} \qquad \frac{}{\vdash_{x^\sigma, y^\tau} \pi_2(x,y) \;=\; y}$$

$$\frac{\vdash_\Delta case_{x,y}(\iota_1 u, s, t) \;=\; s\{u/x\}}{\Delta \vdash u : \sigma_1 \quad \Delta, x^{\sigma_1} \vdash s : \tau \quad \Delta, y^{\sigma_2} \vdash t : \tau}$$

$$\frac{\vdash_\Delta case_{x,y}(\iota_2 u, s, t) \;=\; t\{u/y\}}{\Delta \vdash u : \sigma_2 \quad \Delta, x^{\sigma_1} \vdash s : \tau \quad \Delta, y^{\sigma_2} \vdash t : \tau}$$

$$\frac{\vdash_{\Delta,z} iter_x(s,t)(succ\, z) = t\{iter_x(s,t)\, z/x\}}{\Delta \vdash s : \tau \quad \Delta, x^\tau \vdash t : \tau}\, z^{\mathrm{N}} \text{ not in } \Delta$$

$$\frac{\vdash_\Delta iter_x(s,t)\, 0 = s}{\Delta \vdash s : \tau \quad \Delta, x^\tau \vdash t : \tau} \qquad \frac{\vdash_\Delta (fold_{z,x}(s,t))[\,] \;=\; s}{\Delta \vdash s : \sigma \quad \Delta, z^\tau, x^\sigma \vdash t : \sigma}$$

$$\frac{\vdash_{\Delta,u,v} (fold_{z,x}(s,t))(u :: v) \;=\; t\{(fold_{z,x}(s,t)\, v)/x\}\{u/z\}}{\Delta \vdash s : \sigma \quad \Delta, z^\tau, x^\sigma \vdash t : \sigma}\, v^{\tau^*}, u^\tau \text{ not in } \Delta$$

η Equalities

$$\frac{}{\vdash_{x:1} x = *} \qquad \frac{}{\vdash_{x^{\sigma\times\tau}} (\pi_1 x, \pi_2 x) \;=\; x} \qquad \frac{\vdash_\Delta \lambda x.(tx) \;=\; t}{\Delta \vdash t : \sigma \Rightarrow \tau}\, x^\sigma \text{ not in } \Delta$$

$$\frac{\vdash_{\Delta, z^{\sigma_1+\sigma_2}} case_{x,y}(z, h\{\iota_1 x/z\}, h\{\iota_2 y/z\}) \;=\; h}{\Delta, z^{\sigma_1+\sigma_2} \vdash h : \tau}\, x^{\sigma_1}, y^{\sigma_2} \text{ not in } \Delta$$

ξ Equalities

$$\frac{\vdash_\Delta \lambda x.s \;=\; \lambda x.t}{\vdash_{\Delta,x} s \;=\; t} \qquad \frac{\vdash_\Delta case_{x,y}(u,s,t) = case_{x,y}(u,s',t')}{\Delta \vdash u : \sigma_1 + \sigma_2 \quad \vdash_{\Delta,x^{\sigma_1}} s = s' \quad \vdash_{\Delta,y^{\sigma_2}} t = t'}$$

$$\frac{\vdash_\Delta iter_x(s,t) = iter_x(s,t')}{\Delta \vdash s : \tau \quad \vdash_{\Delta,x^\tau} t = t'} \qquad \frac{\vdash_\Delta fold_{z,x}(s,t) = fold_{z,x}(s,t')}{\Delta \vdash s : \sigma \quad \vdash_{\Delta,z^\tau,x^\sigma} t = t'}$$

Figure 3–5: Equality Axioms of $\mathcal{B}$ for Standard Data Types

Remark: Strictly speaking, the observation assumes that equality is systematically decorated with types. In Figures 3–2 and 3–5 this additional typing information is suppressed for better legibility.

Now the base logic is in place. It is noted that the extensions laid down in this section are meant only as minimal requirements on $\mathcal{B}$ for which the construction of lax logic will be possible. Thus, further data types, logical connectives, and rules may be added to the base logic whenever a specific hardware verification problem requires us to do so. Of course, this is sensible only to the extent that $\mathcal{B}$ remain consistent.

Theorem 3.1.6 *The base logic $\mathcal{B}$, as introduced so far, is consistent.*

Proof: (Idea) The simplest proof is the one by models: Consider a classical set-theoretic model for $\mathcal{B}$ that interprets the ordinary data types in the standard way, in particular such that $[\![\sigma \Rightarrow \tau]\!]$ is the set of all functions from $[\![\sigma]\!]$ to $[\![\tau]\!]$, $[\![\mathbf{N}]\!]$ are the standard natural numbers, and finally such that the type Ω is interpreted by the set $[\![\Omega]\!] = \{true, false\}$. To every closed term $s : \tau$ is assigned an element $[\![s]\!] \in [\![\tau]\!]$ in the usual way, such that $[\![s =_\tau t]\!] = true$ *iff* $[\![s]\!]$ equals $[\![t]\!]$ in the model, *i.e.* are identical elements in $[\![\tau]\!]$. Further, it is shown that $\mathcal{B}$ is correct for this interpretation, *viz.* that if $\vdash \phi$ is derivable, then $[\![\phi]\!] = true$. Consistency of $\mathcal{B}$ then follows from the fact that 0 and $succ(0)$ are different natural numbers in the standard model, *i.e.* $\vdash 0 = succ(0)$ cannot be derivable. ∎

It is well known that higher-order logic is incomplete *wrt.* the standard set-theoretic notion of model, *i.e.* models which interpret a type $\sigma \Rightarrow \tau$ as the set of all functions from σ to τ. One has to consider non-standard models in order to capture the expressibility of a particular higher-order logic in terms of models [Hen50]. Consequently, in contrast to the first-order case there is no canonical system of higher-order logic. Rather, there are quite a number of different formulations[3] in the literature, such as Church's Simple Theory of Types (STT)

[3]The names STT, ITT, and LPE are not introduced in the literature. They are used in

[Chu40], Andrews' systems $\mathcal{F}^{\omega}$ and $\mathcal{Q}_0$ [And86], Coquand's Calculus of Constructions (CC) [CH88], Lambek and Scott's Intuitionistic Type Theory (ITT) [LS86], Fourman and Scott's Logic of Partial Elements (LPE) [Fou89], and many more. In general, one would expect as many different versions of higher-order logics as there are intended notions of models.

Our formulation of $\mathcal{B}$ provides yet another version, which does not compare easily to the systems mentioned above. It basically should be understood as extending many-sorted first-order logic by equality, inductive data types like natural numbers and lists, and a sort Ω of propositions. It is important to point out that Ω has not been added with the aim to make the logic higher-order but to have a particular notion of constraint (*viz.* constraint = list of propositions) available as a type within the logic. For a simpler notion of constraint (such as: constraint = boolean) first-order logic would be enough, although, then the logical connectives *false*, $\wedge$, $\vee$, $\exists$ have to be introduced explicitly. Consequently, the models of $\mathcal{B}$ which we will consider in Section 5 essentially are first-order models with additional structure.

Of the higher-order systems mentioned above the one that is closest to $\mathcal{B}$ is ITT of Lambek and Scott, in the sense that it is many-sorted, explicitly axiomatizing data types, proof irrelevant, and based on the intuitionistic logical primitives $\supset, \forall$. What are some differences between $\mathcal{B}$ and ITT, then? Firstly, the type system of ITT is based on the notion of sets rather than functions, *i.e.* the central feature is a power type operator $\mathbf{P}\tau$ while $\mathcal{B}$ has the function type operator $\tau \Rightarrow \sigma$. To compare both systems one may take the type $\mathbf{P}\tau$ of ITT as an abbreviation for the type $\tau \Rightarrow \Omega$, and the formation of sets $\{x \in A \mid \phi(x)\}$ and elementship $x \in S$ stand for lambda abstraction $\lambda x.\ \phi$ and application $S\,x$, respectively. Other systems based on functional types are STT, $\mathcal{Q}_0$, CC. A second difference is that in ITT, as well as in most other systems of higher-order logic, equality is defined as Leibnitz equality while in $\mathcal{B}$ it is a primitive relation.

---

the following paragraphs only for the purpose of reference.

Thus, the meaning of equality in $\mathcal{B}$ can be chosen freely within certain limits while it is fixed in ITT. Note, though, that in $\mathcal{B}$ we can prove

$$\vdash s = t \cong \forall P.\, P\, s \supset P\, t.$$

Also, in ITT, as well as in most other systems of higher-order logic, equality of propositions is identified with logical equivalence, *i.e.*

$$\Gamma \vdash \phi =_\Omega \psi \quad \textit{iff} \quad \Gamma \vdash \phi \cong \psi$$

while in $\mathcal{B}$ the two ways of comparing propositions are independent concepts. This has the advantage that in $\mathcal{B}$, equality may be given a computationally meaningful interpretation, *e.g.* a decidable relation on types like Ω and Ω^*. How this can be used will be seen from the examples in Chapter 4. A third, less important, difference between $\mathcal{B}$ and ITT is that $\mathcal{B}$ not only axiomatizes the data types products and natural numbers but also sums and lists which are not incorporated in ITT.

To sum up this discussion we claim that if the extensionality axioms

$$\forall f, g.\, (\forall x.\, (f\, x \supset g\, x) \wedge (g\, x \supset f\, x)) \supset f = g$$

are added to $\mathcal{B}$, where f, g have type $\tau \Rightarrow \Omega$, τ arbitrary type, as well as the Peano axioms, then, modulo the canonical translation hinted at before, all theorems of ITT can be derived in $\mathcal{B}$.

The remainder of this section discusses a few basic constructions in $\mathcal{B}$ that will be used later on. First, we state without proof that from iteration *iter* and products a primitive recursion operator $natrec : (\tau \times (\mathbf{N} \Rightarrow \tau \Rightarrow \tau)) \Rightarrow \mathbf{N} \Rightarrow \tau$ can be obtained that satisfies the equations

$$\begin{aligned} natrec(a, f)\, 0 &= a \\ natrec(a, f)\, (succ(n)) &= f\, n\, (natrec(a, f)\, n). \end{aligned}$$

For instance, *natrec* can be defined as follows:

$$natrec \stackrel{df}{=} \lambda z.\, \lambda x.\, \pi_2(iter_y\, ((0, z_1), (succ(y_1), z_2\, y_1\, y_2))\, x)$$

where z, x, y are variables of types $\tau \times (\mathrm{N} \Rightarrow \tau \Rightarrow \tau)$, N, $\mathrm{N} \times \tau$, and z_i, y_i, $i = 1, 2$, abbreviate the terms $\pi_i z, \pi_i y$, respectively. The key to verifying that *natrec* obeys the equations above is to use the induction rule *NatInd* to prove that $\pi_1(\mathit{iter}_y(\cdots, \cdots))$ is the successor function. Similarly, one can construct from the operator *fold* for lists a recursion combinator

$$\mathit{listrec} : (\sigma \times (\tau \Rightarrow \tau^* \Rightarrow \sigma \Rightarrow \sigma)) \Rightarrow \tau^* \Rightarrow \sigma$$

for every choice of types σ, τ for which the equations

$$\begin{aligned} \mathit{listrec}(a, f)\,[\,] &= a \\ \mathit{listrec}(a, f)\,(x :: z) &= f\,x\,z\,(\mathit{listrec}(a, f)\,z) \end{aligned}$$

can be proven in $\mathcal{B}$.

As has been hinted at in the beginning of this section one of the main reasons for extending the base logic as described is that now the notion of constraint in the sense in which we are going to use it is internal to the logic. We can formulate as a proposition of $\mathcal{B}$ the meta-logical statement that some proposition is provable under a constraint. Specifically, we can implement the general approach which is to take as constraints the lists $c = [\gamma_n, \ldots, \gamma_1]$ of propositions, and to say that a proposition ϕ is 'provable under' c *iff* the sequent $\gamma_1, \ldots, \gamma_n \vdash \phi$ is derivable in $\mathcal{B}$. Under this interpretation constraints are in one-one correspondence with terms c of type Ω^* and ϕ is provable under c *iff* the proposition ϕ^c is provable, where ϕ^c is defined for all c by the clauses

$$\phi^{[\,]} = \phi \qquad \phi^{\gamma :: c} = \gamma \supset (\phi^c).$$

Moreover, we can define the map $c, \phi \mapsto \phi^c$ as the following well-formed term of type $\Omega^* \Rightarrow (\Omega \Rightarrow \Omega)$:

$$\mathit{weak} \stackrel{\mathit{df}}{\equiv} \lambda c.\, \lambda \phi.\, (\mathit{fold}_{z,x}\,(\phi, z \supset x))\,c$$

where c is a variable of type Ω^* and ϕ, z, x distinct variables of type Ω. The properties of *weak* are summed up in

Lemma 3.1.7 *Let ϕ, ψ, γ be variables of type Ω and c variable of type Ω^*.*

- *The term 'weak' as defined above is a closed well-formed term of type $\Omega^* \Rightarrow (\Omega \Rightarrow \Omega)$ for which the equations*

$$\vdash_{\phi} \quad weak\,[]\,\phi = \phi$$
$$\vdash_{\gamma,c,\phi} \quad weak\,(\gamma :: c)\,\phi = \gamma \supset (weak\,c\,\phi)$$

  can be derived in $\mathcal{B}$.

- *Proposition $\forall c.\ weak\ c\,\phi$ is provably equivalent to ϕ, and for all $c : \Omega^*$ proposition $weak\ c\,(\phi \supset \psi)$ is provably equivalent to $(weak\,c\,\phi) \supset (weak\,c\,\psi)$, i.e. the sequents*

$$\vdash_{\phi} \quad \phi \cong \forall c.\ weak\,c\,\phi$$
$$\vdash_{\phi,\psi} \quad \forall c.\ weak\,c\,(\phi \supset \psi) \cong (weak\,c\,\phi) \supset (weak\,c\,\psi)$$

  can be derived in $\mathcal{B}$.

Proof: The first part follows from the definition of *weak*, and from β-equality for λ-application and *fold*. The second part is obtained by induction on c and by using the first part of the lemma. ∎

In the following we will take the notation ϕ^c to stand for $weak\,c\,\phi$. Another operation that we will need is concatenation $@ : \tau^* \Rightarrow (\tau^* \Rightarrow \tau^*)$ of lists:

$$@ \stackrel{df}{=} \lambda l.\ \lambda m.\ (fold_{z,x}\,(l, z :: x))\,m$$

where l, m, x are distinct variables of type τ^*, z variable of type τ. For convenience we will use the infix notation $l @ m$ rather than $@\,l\,m$.

Lemma 3.1.8

- *The term $@$ as defined above is a closed well-formed term of type $\tau^* \Rightarrow (\tau^* \Rightarrow \tau^*)$ for every[4] type τ for which the equations*

$$\vdash_{l} \quad l @ [] = l$$
$$\vdash_{z,l,m} \quad l @ (z :: m) = z :: (l @ m)$$

[4]It is assumed that the variables z, x, l, m in the definition of $@$ are chosen in some canonical way for each type τ.

can be derived in $\mathcal{B}$, where l, m are variables of type τ^* and z variable of type τ.

- *The following equations*

$$\begin{aligned} &\vdash_l && [\,] @ m = m \\ &\vdash_{k,l,m} && (k @ l) @ m = k @ (l @ m) \\ &\vdash_{\phi,l,m} && (\phi^l)^m = \phi^{l @ m} \end{aligned}$$

can be derived in $\mathcal{B}$ where ϕ, k, l, m are variables of the appropriate types.

Proof: The first part follows from the definition of $@$, and from β equality for λ and *fold*. The second part is obtained by induction on m, using the first part of the Lemma and of Lemma 3.1.7. ∎

The second part of Lemma 3.1.7 and 3.1.8 can be paraphrased more compactly by stating that the triple $(\Omega^*, [\,], @)$ together with the mapping $c, \phi \mapsto \phi^c$ is a (monoidal) action on Ω that preserves implication. As explained at the beginning of this chapter these data are taken to define a notion of constraint.

Definition 3.1.9 *A* notion of constraint *for $\mathcal{B}$ is a triple $(\mathbf{C}, 1, \cdot)$, where $\mathbf{C}$ is a type and $1 : \mathbf{C}$, $\cdot : \mathbf{C} \times \mathbf{C} \Rightarrow \mathbf{C}$ closed terms, together with a map $c, \phi \mapsto \phi^c$ that assigns to each term $c : \mathbf{C}$ and proposition $\phi : \Omega$ a proposition $\phi^c : \Omega$, subject to the following conditions:*

- *$(\mathbf{C}, 1, \cdot)$ is an internal monoid, i.e. the monoid equations $1 \cdot c = c = c \cdot 1$ and $(c \cdot d) \cdot e = c \cdot (d \cdot e)$ can be derived in $\mathcal{B}$.*
- *$(\mathbf{C}, 1, \cdot)$ is an action on formulae, i.e. $\phi^1 = \phi$ and $(\phi^c)^d = \phi^{c \cdot d}$.*
- *The action preserves entailment, i.e. a sequent $\Gamma, \phi_1, \ldots, \phi_n \vdash_\Delta \phi$ is derivable in $\mathcal{B}$ iff for all $\Delta \vdash c : \mathbf{C}$ the sequents $\Gamma, {\phi_1}^c, \ldots, {\phi_n}^c \vdash_\Delta \phi^c$ are derivable.*
- *The action respects substitution, i.e. if x is a variable and t a term of type τ, then $(\phi^c)\{t/x\} = (\phi\{t/x\})^{c\{t/x\}}$.*

Remark: The third condition covers the special case $n = 0$, *viz.* $\Gamma \vdash_\Delta \phi$ *iff* $\Gamma \vdash_\Delta \phi^c$ for all $\Delta \vdash c : \Omega$. For $\Gamma = \phi$ this means $\phi \vdash_\Delta \phi^c$ for all $\Delta \vdash c : \Omega$. Note, the definition does not require that the mapping $c, \phi \mapsto \phi^c$ is actually expressible within the logic as a term of type $\mathbf{C} \times \Omega \Rightarrow \Omega$, as is the case with the notion of constraint $(\Omega^*, [\,], @)$ considered later. It may also be a syntactic translation. The fourth condition is there to make sure that this syntactic translation is well-behaved *wrt.* substitution. We remark that for an action on truth-values rather than formulae one would replace in the second condition the equality by equivalence. This, too, would suffice for our purposes.

Lemma 3.1.10 $(\Omega^*, [\,], @)$ *together with* $\phi^c \equiv \mathit{weak}\, c\, \phi$ *is a notion of constraint.*

Proof: The first two properties of Definition 3.1.9 follow immediately from Lemmata 3.1.7, 3.1.8. The *if* direction of the second condition is trivial anyway since $\phi^1 = \phi$. The *only-if* part is proven by induction on the number n of hypotheses and the second part of Lemma 3.1.7. The fourth property is satisfied trivially by $(\Omega^*, [\,], @)$ since $\phi^c = \mathit{weak}\, c\, \phi$, where *weak* is a closed term. ∎

In connection with using a list $c : \Omega^*$ of propositions as a constraint a function $\sqcap : \Omega^* \Rightarrow \Omega$ will be useful that conjoins all the propositions in c into a single proposition $\sqcap c$. This map can be defined by

$$\sqcap \stackrel{df}{\equiv} \mathit{fold}_{z,x}\,(\mathit{true}, z \wedge x).$$

Lemma 3.1.11 *The propositions* ϕ^c *and* $(\sqcap c) \supset \phi$ *are provably equivalent, i.e. the sequent*

$$\vdash_{\phi,c} \phi^c \cong (\sqcap c) \supset \phi$$

is derivable in $\mathcal{B}$*, where* ϕ *and* c *are variables of type* Ω *and* Ω^**, respectively.*

Proof: easy, by list induction on c. ∎

3.1.3 Lax Logic

We are now going to extend $\mathcal{B}$ by a modal operator $\Diamond$ with the intended meaning that $\Diamond\phi$ should be true if there is a constraint c such that ϕ holds under c. Given a notion of constraint $(\mathbf{C}, 1, \cdot)$, this appears to suggest

$$\Diamond\phi \stackrel{\mathit{df}}{=} \exists c^{\mathbf{C}}.\, \phi^c \tag{3.1}$$

so that lax logic simply becomes a definitional extension of $\mathcal{B}$. Let us see why this is not a good idea. Note first, however, that the rules

$$\frac{\phi \vdash_\Delta \Diamond\phi}{}\Diamond I \qquad \frac{\Gamma, \Diamond\phi \vdash_\Delta \Diamond\psi}{\Gamma, \phi \vdash_\Delta \psi}\Diamond F$$

indeed are derived rules under this definition of $\Diamond$; $\Diamond I$ follows from the equivalence $\phi^1 \cong \phi$, and $\Diamond F$ is a consequence of the fact that $\phi \mapsto \phi^c$ preserves entailment. The problem with the above naive definition (3.1) comes with the rule scheme

$$\frac{\Diamond\Diamond\phi \vdash_\Delta \Diamond\phi}{}\Diamond M$$

which translates to the sequent $\exists c.\, (\exists d.\, \phi^d)^c \vdash_\Delta \exists d.\, \phi^d$ which is interderivable with the axiom scheme

$$\forall c.\, ((\exists d.\, \phi^d)^c \supset \exists d.\, \phi^d).$$

Now, suppose this were derivable, then we could conclude that for all $\Delta \vdash c : \mathbf{C}$ the proposition $(\exists d.\, \phi^d)^c$ is provable *iff* $\exists d.\, \phi^d$ is provable. But this says that no constraint has any influence on the provability of a proposition of form $\exists d.\, \phi^d$ which appears to be a strong condition to impose on a notion of constraint.

A second problem with the definition (3.1) arises for the particular notion of constraint $(\Omega^*, [\,], @)$. Here, ϕ^c is equivalent to $\Box c \supset \phi$. We can take $c = [\phi]$ as the constraint to get

$$\phi^{[\phi]} \cong (\Box[\phi]) \supset \phi \cong \phi \supset \phi \cong \mathit{true}.$$

So, for every ϕ there is a proof of $\exists c.\, \phi^c$. Hence under definition (3.1) any proposition of form $\Diamond\phi$ would be provable, which trivializes $\Diamond$ in a very undesirable way.

So, we better not hard-wire this property of the particular notion of constraint $(\Omega^*, [\,], @)$ into the logic.

The third and most important reason for not adopting definition (3.1) is that there is no guarantee, that from a proof of $\Diamond\phi = \exists c.\, \phi^c$ we can actually extract a constraint c and a proof of ϕ^c. Since $\mathcal{B}$ can be an arbitrarily strong classical logic this may actually be impossible, in general.

Instead of using a definition like (3.1) we are going to embed $\mathcal{B}$ properly within an intuitionistic first-order logic containing $\Diamond$ as a primitive modal operator together with associated rules $\Diamond I$, $\Diamond M$, and $\Diamond F$. Apart from containing $\mathcal{B}$ as a sub-logic it will be equipped with its own first-order connectives. This 'top-up' logic will be called *lax logic* and denoted by the symbol $\mathcal{L}$.

The collection of types and terms of $\mathcal{L}$, as well as the typings, are simply inherited wholesale from the base logic. The embedding $\mathcal{B} \hookrightarrow \mathcal{L}$ will be witnessed syntactically by an operator ι that promotes a proposition ϕ from $\mathcal{B}$ into a formula $\iota\phi$ of $\mathcal{L}$. This mapping ι will be shown to preserve and reflect provability, *i.e.* ϕ is provable in $\mathcal{B}$ *iff* $\iota\phi$ is provable in $\mathcal{L}$ (Theorem 3.1.16).

FORMULAE. The *formulae* of $\mathcal{L}$, ranged over by the meta-variables M, N, K, *etc.* are given according to the grammar

$$\begin{aligned} M \;::=\; & \mathit{true} \mid M \wedge M \mid \mathit{false} \mid M \vee M \mid M \supset M \mid \\ & \forall x.\, M \mid \exists x.\, M \mid \Diamond M \mid \iota\phi \end{aligned}$$

where ϕ ranges over propositions of $\mathcal{B}$ and x over variables. These are the only formulae of $\mathcal{L}$. These connectives are to be distinguished from the primitive connectives $\supset$, $\forall$, and the abbreviations $\mathit{true}, \mathit{false}, \vee, \exists$ defined for $\mathcal{B}$. However, as it is clear from the context within a formula in which a symbol occurs whether it belongs to $\mathcal{B}$ or $\mathcal{L}$, namely inside or outside the scope of ι, we may use the same symbol in both cases. For instance, in

$$\exists x.\, (M \wedge \iota(\phi \wedge \psi))$$

the $\exists$ and the left $\wedge$ are primitive connectors in $\mathcal{L}$ while the right $\wedge$ is the abbreviation of conjunction in terms of $\supset$ and $\forall$ in $\mathcal{B}$. The well-formed formulae M of $\mathcal{L}$ together with the set $FV(M)$ of *free* variables are defined as follows: A formula M is *well-formed* if

- M is *true* or *false*; $FV(M) = \emptyset$,
- M is one of $M_1 \wedge M_2$, $M_1 \vee M_2$, or $M_1 \supset M_2$ and both M_1 and M_2 are well-formed formulae; $FV(M) = FV(M_1) \cup FV(M_2)$,
- N is a well-formed formula and M is $\Diamond N$; $FV(M) = FV(N)$,
- M is one of $\forall x.\, N$ or $\exists x.\, N$ where x is a variable and N a well-formed formula; $FV(M) = FV(N) \setminus \{x\}$,
- M is $\iota\phi$ with ϕ a well-formed proposition of $\mathcal{B}$; $FV(M) = FV(\phi)$.

$$\frac{}{\Delta \vdash \mathit{true}\ \mathbf{wff}} \qquad \frac{}{\Delta \vdash \mathit{false}\ \mathbf{wff}} \qquad \frac{\Delta \vdash \iota\phi\ \mathbf{wff}}{\Delta \vdash \phi : \Omega} \qquad \frac{\Delta \vdash \Diamond M\ \mathbf{wff}}{\Delta \vdash M\ \mathbf{wff}}$$

$$\frac{\Delta \vdash M \wedge N\ \mathbf{wff}}{\Delta \vdash M\ \mathbf{wff} \quad \Delta \vdash N\ \mathbf{wff}} \qquad \frac{\Delta \vdash \forall x.\, M\ \mathbf{wff}}{\Delta, x \vdash M\ \mathbf{wff}} \qquad \frac{\Delta \vdash \exists x.\, M\ \mathbf{wff}}{\Delta, x \vdash M\ \mathbf{wff}}$$

$$\frac{\Delta \vdash M \supset N\ \mathbf{wff}}{\Delta \vdash M\ \mathbf{wff} \quad \Delta \vdash N\ \mathbf{wff}} \qquad \frac{\Delta \vdash M \vee N\ \mathbf{wff}}{\Delta \vdash M\ \mathbf{wff} \quad \Delta \vdash N\ \mathbf{wff}}$$

$$\frac{\Delta, x \vdash M\ \mathbf{wff}}{\Delta \vdash M\ \mathbf{wff}}\ (x\ \textit{variable, not in}\ \Delta) \qquad \frac{\Delta, y, x, \Delta' \vdash M\ \mathbf{wff}}{\Delta, x, y, \Delta' \vdash M\ \mathbf{wff}}$$

Figure 3–6: Well-Formed Formulae of Lax Logic

The reader will notice that the only possibility for a formula not to be well-formed is that it contains a sub-formula $\iota\phi$ where ϕ is not a well-formed proposition of

$\mathcal{B}$. A well-formed formula M is *closed* if $FV(M) = \emptyset$. Again we remark that free variables must occur as sub-terms. The notion of a *bound* variable is analogous to $\mathcal{B}$. As in the case of terms we can formalize the notion of a well-formed formula using a new judgement

$$\Delta \vdash M \text{ wff}$$

where $\Delta = x_1, \ldots x_n$ is a context. The rules are shown in Figure 3–6.

Observation 3.1.12

- *A formula M is well-formed with all its free variables in context Δ iff $\Delta \vdash M$ wff is derivable by the rules of Figure 3–6.*
- *If $\Delta, x^\tau \vdash M$ wff and $\Delta \vdash t : \tau$, then $\Delta \vdash M\{t/x\}$ wff.*

Equivalence $\cong$ of formulae is defined as before, *viz.* $M \cong N \stackrel{\text{df}}{=} M \supset N \wedge N \supset M$.

$\mathcal{L}$-INFERENCES. The calculus of entailment between formulae of $\mathcal{L}$ is presented via the judgement

$$\Gamma \vdash_\Delta M$$

called an *$\mathcal{L}$-sequent* to distinguish it from sequents of $\mathcal{B}$, which consequently will be referred to as $\mathcal{B}$-sequents. We will simply talk about *sequents* when the logical system is clear from the context. $\Gamma = M_1, \ldots, M_n$ is a list of *hypotheses* and M the *assertion* of the sequent. All M_i and M must be well-formed formulae with free variables in Δ. The inference rules for deriving $\mathcal{L}$-sequents are listed in Figure 3–7 (structural rules), Figure 3–9 (logical rules), and Figure 3–8 (induction rules). They are as in standard first-order logic plus the embedding rule ι, and the three special rules $\Diamond I$, $\Diamond M$, and $\Diamond F$ describing the properties of $\Diamond$. As remarked at the beginning of this chapter these rules reflect the basic intended properties of proving a formula 'under a constraint', *i.e.* the properties of a notion of constraint. The next section (3.2) will show how the intuitive interpretation of these rules can be made precise.

$$\frac{\Gamma, M \vdash_\Delta N}{\Gamma \vdash_\Delta N \quad \Delta \vdash M \text{ wff}}\, weak \qquad \frac{\Gamma \vdash_{\Delta, x} M}{\Gamma \vdash_\Delta M}\, weak \ (x \text{ not in } \Delta)$$

$$\frac{M \vdash_\Delta M}{\Delta \vdash M \text{ wff}}\, id$$

$$\frac{\Gamma \vdash_{\Delta, y, x, \Delta'} M}{\Gamma \vdash_{\Delta, x, y, \Delta'} M}\, perm \qquad \frac{\Gamma, N, M, \Gamma' \vdash_\Delta K}{\Gamma, M, N, \Gamma' \vdash_\Delta K}\, perm$$

$$\frac{\Gamma\{t/x\} \vdash_\Delta M\{t/x\}}{\Gamma \vdash_{\Delta, x:\tau} M \quad \Delta \vdash t : \tau}\, sub \qquad \frac{\Gamma \vdash_\Delta N}{\Gamma \vdash_\Delta M \quad \Gamma, M \vdash_\Delta N}\, cut$$

Figure 3–7: Structural Rules of Lax Logic

$$\frac{\Gamma \vdash_\Delta \forall z.\, M}{\Gamma \vdash_\Delta M\{[]/z\} \quad \Gamma, M \vdash_{\Delta, x, z} M\{x :: z/z\}}\, ListInd\ (x, z \text{ of type } \tau, \tau^*)$$

$$\frac{\Gamma \vdash_\Delta \forall z.\, M}{\Gamma \vdash_\Delta M\{0/z\} \quad \Gamma, M \vdash_{\Delta, z} M\{succ\, z/z\}}\, NatInd\ (z \text{ of type } \mathbf{N})$$

Figure 3–8: Induction Rules of Lax Logic

Remark: It may appear that rules $\exists E$, $\vee E$, in Figure 3–9 should be stated in a more general form by adding an arbitrary list of hypotheses Γ on both sides of the rule bar, plus the condition that x not be free in Γ. However, this is not necessary as any hypotheses can be moved out of the way to the assertion side using rules $\supset I$ and $\supset E$.

Observation 3.1.13 *Every inference $\Gamma \vdash_\Delta M$ derived by means of rules of Figures 3–7, 3–9, 3–8 is well-formed, i.e. Δ is a context (non-duplicating list of variables), and the assertion M as well as all hypotheses in Γ are well-formed formulae in context Δ.*

Remark: This observation means that we do not need to add the usual side-conditions on variables x, z in the induction rules of Figure 3–8 to make sure that Γ does not depend on x and also in rule *ListInd* that M does not depend on x.

If the premisses of rule *ListInd* have been derived, then we know from the right sub-derivation that Δ, x, z is a context, whence x, z do not occur in Δ. From the left sub-derivation we obtain that $\Gamma, M\{\emptyset/z\}$ are well-formed in context Δ, whence x, z cannot be free in Γ and x cannot be free in M. A similar remark applies to rule *NatInd*.

The reader will notice that there is no equality in lax logic, *i.e.* there are no atomic formulae like $s = t$. This is so for good reasons: Equality can be lifted from the base logic, the formulae $\iota(s = t)$ can be shown to behave like an equality in lax logic.

Lemma 3.1.14 *The following equality substitution schema*

$$\frac{\Gamma \vdash_\Delta M\{s/x\}}{\Delta, x^\tau \vdash M \text{ wff} \quad \Gamma \vdash_\Delta \iota(s =_\tau t) \quad \Gamma \vdash_\Delta M\{t/x\}}\ \textit{Subst}$$

is a derived rule of lax logic.

Proof: By induction on the structure of formula M. The crucial base case is when $M = \iota\phi$ for which the schema follows from substitution in the base logic and the embedding rule ι. All the other cases are obtained by first deconstructing the top-level connective in M, applying the inductional hypothesis to its components, and then reconstructing M with the appropriate rules. ∎

Let us point out two central properties of $\Diamond$:

Theorem 3.1.15

- *In the context of the other rules, $\Diamond M$ and $\Diamond F$ are equivalent to the inference rule:*

$$\frac{\Gamma, \Diamond M \vdash_\Delta \Diamond N}{\Gamma, M \vdash_\Delta \Diamond N}\ \Diamond L$$

i.e. from $\Diamond M$ and $\Diamond F$ one can derive $\Diamond L$ and vice versa.

- $\Diamond$ *is strongly extensional, i.e. the inference rule*

$$\frac{\vdash_\Delta (M \cong N) \supset (\Diamond M \cong \Diamond N)}{\Delta \vdash M \ \mathbf{wff} \quad \Delta \vdash N \ \mathbf{wff}} \Diamond ext$$

is derivable.

Proof: In Figure 3–13 of the next section (Section 3.2) a derivation of $\Diamond L$ from rules $\Diamond F$ and $\Diamond M$ is given. The derivation of both $\Diamond F$ and $\Diamond M$ from $\Diamond L$, as well as the derivation of $\Diamond ext$ is easy and left to the reader. ∎

The base logic $\mathcal{B}$ is contained within the ι fragment — the class of formulae of shape $\iota\phi$ — of $\mathcal{L}$ via the embedding rule ι. Thus, $\mathcal{L}$ can be viewed as an extension of $\mathcal{B}$. It is very desirable that this extension do not change the deductive properties of $\mathcal{B}$ so that new theorems about $\mathcal{B}$ would be provable in $\mathcal{L}$. The following theorem says that all $\iota\phi$ which are provable in the extension $\mathcal{L}$ are provable in $\mathcal{B}$ already, or that $\mathcal{B}$ corresponds exactly to the ι fragment of $\mathcal{L}$.

Theorem 3.1.16 (Conservativity) *$\mathcal{L}$ is conservative over $\mathcal{B}$, i.e. if some $\iota\phi$ is derivable in $\mathcal{L}$, then ϕ must already be a theorem of $\mathcal{B}$.*

This perhaps is not surprising if we bring to mind the rules of $\mathcal{L}$ for $\Diamond$. $\Diamond I$, $\Diamond F$, and $\Diamond M$ all are one-way rules for $\Diamond$: once $\Diamond$ is introduced there is no way to eliminate it again. So, proofs of a proposition $\iota\phi$ in $\mathcal{L}$ cannot be achieved by detouring over modal formulae. Theorem 3.1.16 holds in a more general form as follows:

Theorem 3.1.17 (Strong Conservativity) *Let M be a well-formed formula in context Δ such that $\vdash_\Delta M$ is derivable in $\mathcal{L}$. Then, we must have a derivation of $\vdash_\Delta M'$ in $\mathcal{B}$, where proposition M' is obtained from M by removing all occurrences of $\Diamond$ and ι and replacing the other logical connectives by their respective counterparts in $\mathcal{B}$.*

This theorem is a necessary condition for $\Diamond$ to act as a place-holder: it must not be possible to turn a non-theorem of $\mathcal{B}$ into a theorem of $\mathcal{L}$ merely by introducing $\Diamond$s in certain places.

Proof: By induction on the structure of derivations one proves that every derivation of a $\mathcal{L}$-sequent $M_1, \ldots, M_k \vdash_\Delta M$ can be translated into a derivation of the $\mathcal{B}$-sequent $M'_1, \ldots, M'_k \vdash_\Delta M'$. The crucial cases are the rules ι, $\Diamond I$, $\Diamond M$, and $\Diamond F$, which are all trivialized by the translation. For instance, sequent $\Diamond\Diamond M \vdash_\Delta \Diamond M$ translates into $M' \vdash_\Delta M'$. ∎

As a corollary to Theorem 3.1.16 we conclude that $\mathcal{L}$ is consistent.

Theorem 3.1.18 *$\mathcal{L}$ is consistent.*

Proof: The theorem follows from consistency of $\mathcal{B}$: If $\vdash$ *false* were derivable in $\mathcal{L}$, then also $\vdash \iota$ *false* would be derivable because of rule *falseE*. But by Theorem 3.1.16 this implies that $\vdash$ *false* is derivable in $\mathcal{B}$. Contradiction. ∎

We have seen that we can turn a theorem of $\mathcal{L}$ into a theorem of $\mathcal{B}$ by removing all $\Diamond$s and ιs. What about the other direction? Surely, we do not want to have that a theorem of $\mathcal{B}$ becomes a theorem of $\mathcal{L}$ by arbitrarily introducing $\Diamond$s and ιs for this would mean that $\mathcal{L}$ is a trivial extension. In fact, this is not the case. It will be shown below (*cf.* Lemma 3.1.19) that if $\Diamond\iota\phi \supset \iota\phi$ is a theorem of $\mathcal{L}$, then also $\iota\phi$ must be a theorem of $\mathcal{L}$, whence ϕ must be a theorem of $\mathcal{B}$. So, for a non-theorem ϕ in $\mathcal{B}$, which must exist by consistency, we have that $\Diamond\iota\phi \supset \iota\phi$ is not provable in $\mathcal{L}$, while, of course $\phi \supset \phi$ is provable in $\mathcal{B}$. Thus, the other direction of Theorem 3.1.17 is not true in general and $\mathcal{L}$ is a non-trivial extension of $\mathcal{B}$.

Lemma 3.1.19 *Let M be a well-formed formula in context Δ such that $\vdash_\Delta M$ is derivable in $\mathcal{L}$. Then, there is a derivation of $\vdash_\Delta M_0$ in $\mathcal{L}$, where formula M_0 is obtained from M by replacing all those sub-formulae by 'true' that are prefixed with $\Diamond$.*

Proof: One shows by induction on the structure of a derivation that whenever a sequent $\Gamma \vdash_\Delta M$ is derivable, then there is a derivation of $\Gamma_0 \vdash_\Delta M_0$. The translation $M \mapsto M_0$ is formally given by the scheme

$$\begin{array}{rclcrcl} \mathit{true}_0 & = & \mathit{true} & & \mathit{false}_0 & = & \mathit{false} \\ (M \wedge N)_0 & = & M_0 \wedge N_0 & & (M \vee N)_0 & = & M_0 \vee N_0 \\ (M \supset N)_0 & = & M_0 \supset N_0 & & (\iota\phi)_0 & = & \iota\phi \\ (\forall x.\, M)_0 & = & \forall x.\, M_0 & & (\exists x.\, M)_0 & = & \exists x.\, M_0 \\ (\Diamond M)_0 & = & \mathit{true}. & & & & \end{array}$$

∎

As a corollary to this lemma we note:

Theorem 3.1.20 *$\mathcal{L} + \forall z^{\Omega}.\, \Diamond\, \iota\, z$ is consistent.*

Proof: If $\forall z.\, \Diamond\, \iota\, z$ were inconsistent, *i.e.* we could derive $\vdash (\forall z.\, \Diamond\, \iota\, z) \supset \mathit{false}$, then by Lemma 3.1.19 we could also derive $\vdash (\forall z.\, \mathit{true}) \supset \mathit{false}$. But this means $\mathcal{L}$ is inconsistent, since $\vdash_s \mathit{true}$ and hence $\vdash \forall z.\, \mathit{true}$ is derivable. Contradiction.

∎

So much for a definition of lax logic in sequent style. In Section 3.3 also a natural deduction presentation is given which will make later example derivations much more economic. Let us finish off this section with highlighting the central features of lax logic.

- In $\mathcal{L}$ we have induction for natural numbers and lists. Being able to construct inductive proofs of formulae which are arbitrarily complex in terms of $\Diamond$-modalities is what fleshes out the bare bones of constraint manipulation, as will be seen from the examples in Chapter 4.

- $\mathcal{L}$ is a *first-order* logic. We cannot quantify over formulae as there is no type of all formulae. This is an important restriction which drastically reduces the complexity of constraint information extracted from proofs in $\mathcal{L}$. In the next section we will see how to extract from every proof of a formula M a constraint

term the type of which, denoted by $|M|$, is determined by the structure of M. Now, if $\mathcal{L}$ were higher-order, then M might have a free variable x that stands for an arbitrary formula. In that case the type $|M|$ will depend on the actual formula substituted in for x, whence $|M|$ would be a *dependent type*. Adding dependent types to the type system of $\mathcal{L}$ and $\mathcal{B}$ is a major complication that we want to avoid in this thesis.

A consequence of the restriction to first-order quantifiers is that the logical connectives *false*, *true*, $\vee, \wedge, \exists$ cannot be defined in terms $\supset$ and $\forall$. They must be introduced explicitly as primitive symbols one by one. Note, however, that since the base logic is part of $\mathcal{L}$ we may still quantify over the type Ω of propositions, though. For instance, the formula $\forall z^{\Omega}.\, \bigcirc \iota z$ is well-formed and will play a prominent role later.

• Finally, $\mathcal{L}$ is intuitionistic, *i.e.* $\mathcal{L}$-sequents allow a single formula on the right side of $\vdash$ and there is no axiom of the excluded middle or an equivalent classical principle. This allows us to extract from derivations in $\mathcal{L}$ constraint information in the form of ordinary lambda terms. Therefore, $\mathcal{L}$ has to be a closed system with no formulae and inference rules other than those defined above. For every new feature that is to be added to $\mathcal{L}$ constraint extraction must be defined and proven correct separately. In contrast, $\mathcal{B}$, which is proof irrelevant, can be arbitrarily strong without any change to the theory of lax logic. This explains also why we distinguish two kinds of sequent $\vdash, \Vdash$ with the consequence that some inference rules are duplicated. We want to keep track of which parts of a proof are relevant for later extracting constraint information and which parts are not.

$$\frac{}{\vdash \mathit{true}}\,\mathit{trueI} \qquad \frac{\mathit{false} \vdash_\Delta M}{\Delta \vdash M\ \mathbf{wff}}\,\mathit{falseE}$$

$$\frac{\Gamma \vdash_\Delta M \wedge N}{\Gamma \vdash_\Delta M \quad \Gamma \vdash_\Delta N}\wedge I \qquad \frac{M \vee N \vdash_\Delta K}{M \vdash_\Delta K \quad N \vdash_\Delta K}\vee E_r$$

$$\frac{M \wedge N \vdash_\Delta M}{\Delta \vdash M\ \mathbf{wff} \quad \Delta \vdash N\ \mathbf{wff}}\wedge E_l \qquad \frac{M \vdash_\Delta M \vee N}{\Delta \vdash M\ \mathbf{wff} \quad \Delta \vdash N\ \mathbf{wff}}\vee I_l$$

$$\frac{M \wedge N \vdash_\Delta N}{\Delta \vdash M\ \mathbf{wff} \quad \Delta \vdash N\ \mathbf{wff}}\wedge E_r \qquad \frac{N \vdash_\Delta M \vee N}{\Delta \vdash M\ \mathbf{wff} \quad \Delta \vdash N\ \mathbf{wff}}\vee I_r$$

$$\frac{\Gamma \vdash_\Delta M \supset N}{\Gamma, M \vdash_\Delta N}\supset I \qquad \frac{\Gamma, M \vdash_\Delta N}{\Gamma \vdash_\Delta M \supset N}\supset E$$

$$\frac{\Gamma \vdash_\Delta \forall x.\, M}{\Gamma \vdash_{\Delta, x} M}\forall I \ \ (x \textit{ not free in } \Gamma) \qquad \frac{\forall x.\, M \vdash_\Delta M\{t/x\}}{\Delta \vdash t : \tau \quad \Delta, x^\tau \vdash M\ \mathbf{wff}}\forall E$$

$$\frac{M\{t/x\} \vdash_\Delta \exists x.\, M}{\Delta \vdash t : \tau \quad \Delta, x^\tau \vdash M\ \mathbf{wff}}\exists I \qquad \frac{\exists x.\, M \vdash_\Delta N}{M \vdash_{\Delta, x} N}\exists E \ \ (x \textit{ not free in } N)$$

$$\frac{M \vdash_\Delta \Diamond M}{\Delta \vdash M\ \mathbf{wff}}\Diamond I \qquad \frac{\Diamond\Diamond M \vdash_\Delta \Diamond M}{\Delta \vdash M\ \mathbf{wff}}\Diamond M \qquad \frac{\Gamma, \Diamond M \vdash_\Delta \Diamond N}{\Gamma, M \vdash_\Delta N}\Diamond F$$

$$\frac{\iota\phi_1, \ldots, \iota\phi_k \vdash_\Delta \iota\psi}{\phi_1, \ldots, \phi_k \vdash_\Delta \psi}\,\iota$$

Figure 3–9: Logical Inference Rules of Lax Logic

3.2 Constraint Extraction

By exploiting the constructive nature of $\mathcal{L}$ as a first-order extension of $\mathcal{B}$ we are going to extract constraint information from derivations in $\mathcal{L}$. We will show how, by eliminating the operator $\Diamond$ in favour of constraints, this information may be analyzed and used to translate back derivations in $\mathcal{L}$ into derivations in $\mathcal{B}$ so that formulae become propositions and $\mathcal{L}$-sequents become $\mathcal{B}$-sequents. Although the extraction will work for any notion of constraint (*cf.* Def. 3.1.9) we will, for the sake of definiteness, focus on the notion of constraint given by the triple $(\Omega^*, [\,], @)$ for which

$$\phi^{[\gamma_n,\ldots,\gamma_1]} \stackrel{df}{=} \mathit{weak}[\gamma_n,\ldots,\gamma_1]\,\phi = \gamma_1 \supset \cdots \supset \gamma_n \supset \phi.$$

For this model a translation process will be defined that allows one to compute for each derivation of an $\mathcal{L}$-sequent $\vdash M$ the derivation of a $\mathcal{B}$-sequent $\vdash M^0$, where M^0 is a proposition obtained from M by replacing all occurrences of $\Diamond$ by certain constraints. So, if $\Diamond(\cdot)$ is an occurrence of $\Diamond$ in M, then it will be replaced by $(\cdot)^c$, where c is a term of type Ω^*. In particular, the translation will be such that a derivation of $\vdash \iota\phi$ is translated into a derivation of $\vdash \phi$.

For simplicity let us restrict attention for a moment to proofs of formulae in the empty context, *i.e.* on derivations of $\mathcal{L}$-sequents $\vdash M$ with M a closed formula. As usual a derivation of a sequent $\vdash \phi$ or $\vdash M$ is called a *proof* of ϕ or M, respectively. Our plan is to associate with every closed M a *constraint type* $|M|$ and a *constraint predicate* $M^* : |M| \Rightarrow \Omega$, such that from every proof of M in $\mathcal{L}$ a closed *constraint term* t of type $|M|$ can be extracted together with a proof of M^*t in $\mathcal{B}$. Thus, we want

$$\begin{aligned}
\mathit{formula}\,M \;&\mapsto\; |M|\ \mathit{type} && (3.2)\\
&\qquad\; M^* : |M| \Rightarrow \Omega\ \mathit{well\text{-}formed\ predicate} && (3.3)\\
\mathit{proof\ of}\ M \;&\mapsto\; t : |M|\ \mathit{well\text{-}formed\ term} && (3.4)\\
&\qquad\; \mathit{proof\ of}\ M^*t. && (3.5)
\end{aligned}$$

Intuitively, $|M|$ is the type of constraint information for M and M^* a predicate telling how M is parameterized or modified by constraint information. In general, M^*x will be a weakening of M depending on constraint term x. The extraction yields for each proof of M in $\mathcal{L}$ some specific constraint term t and a proof of M^*t.

As an example consider the specification of the decrementor:

$$SPEC \equiv \forall n.\, \Diamond\, \iota\, (succ(dec\, n) = n).$$

Here, the constraint type will be

$$|SPEC| \equiv \mathbf{N} \Rightarrow (\Omega^* \times \mathbf{1})$$

and the constraint predicate

$$\begin{aligned} SPEC^*z &\equiv \forall n.\, (succ(dec\, n) = n)^{\pi_1(z\, n)} \\ &\cong \forall n.\, \sqcap(\pi_1(z\, n)) \supset succ(dec\, n) = n. \end{aligned}$$

Further, from a proof of $SPEC$ in $\mathcal{L}$ we obtain a constraint term t of type $\mathbf{N} \Rightarrow (\Omega^* \times \mathbf{1})$ and a proof of $SPEC^*t$ in $\mathcal{B}$. Thus, the proposition $\sqcap(\pi_1(t\, n))$ then is the concrete constraint by which the decrementor's ideal specification has to be weakened to turn the lax proof of $SPEC$ into a proper, 'rigid' proof in $\mathcal{B}$.

For the ordinary first-order connectives of $\mathcal{L}$ the translation process that we will define is known from standard proof extraction techniques for intuitionistic logics, *cf.* [TvD88] for instance. Christine P.-Mohring [PM89] introduces a similar extraction process for the higher-order type theory of the Calculus of Constructions. There, $|M|$ in general is a dependent type while in our first-order setting it is a simple non-dependent type.

We begin with the first part of the translation, *i.e.* with 3.2 and 3.3. The crucial connectives to be covered are $\Diamond$ and ι. Let us consider ι first. Suppose ϕ some proposition of the base logic. Since proofs of $\iota\phi$ in $\mathcal{L}$ are to correspond

one-one to proofs of ϕ in $\mathcal{B}$, and since in $\mathcal{B}$ no notion of proof information is assumed, the translation for $\iota\phi$ is taken to be simply

$$|\iota\phi| \stackrel{df}{=} \mathbf{1} \qquad (\iota\phi)^* \stackrel{df}{=} \lambda z.\phi$$

where z is a fresh variable of type $\mathbf{1}$ not occurring in ϕ. The definition of the constraint predicate $(\iota\phi)^*$ of type $\mathbf{1} \Rightarrow \Omega$ could equally well be written in a point-wise manner, *viz.* as $(\iota\phi)^* z = \phi$. Both equations are equivalent since we are assuming β and η-equalities and the ξ-rule for function spaces. This latter form of definition will be adopted in the sequel.

What is the translation of $\Diamond$? Well, from proofs of $\Diamond\iota\phi$ we seek to extract a term $c : \Omega^*$ and a proof of ϕ^c. Given the translation defined above for ι this can be expressed also by saying that the constraint information obtained from a proof of $\Diamond\iota\phi$ is a pair (c, d) with c a term of type Ω^* and $d : |\iota\phi|$ constraint information for $\iota\phi$ with the property that $\vdash ((\iota\phi)^* d)^c$ is derivable in $\mathcal{B}$. This leads us to define

$$|\Diamond M| \stackrel{df}{=} \Omega^* \times |M| \qquad (\Diamond M)^* z \stackrel{df}{=} (M^*(\pi_2 z))^{\pi_1 z}$$

where z is a fresh variable of type $\Omega^* \times |M|$, not occurring in M already. Again, the 'proper' global definition of the constraint predicate $(\Diamond M)^*$ is obtained from the equation shown by λ-abstraction over z. For the other connectives constraint type and constrained predicates are declared as follows:

$$\begin{array}{rcl@{\qquad}rcl}
|\mathit{true}| & \stackrel{df}{=} & \mathbf{1} & \mathit{true}^* z & \stackrel{df}{=} & \mathit{true} \\
|M \wedge N| & \stackrel{df}{=} & |M| \times |N| & (M \wedge N)^* z & \stackrel{df}{=} & M^*(\pi_1 z) \wedge N^*(\pi_2 z) \\
|\mathit{false}| & \stackrel{df}{=} & \mathbf{0} & \mathit{false}^* z & \stackrel{df}{=} & \mathit{false} \\
|M \vee N| & \stackrel{df}{=} & |M| + |N| & (M \vee N)^* z & \stackrel{df}{=} & (\exists x^{|M|}.\, z = \iota_1 x \wedge M^* x) \\
 & & & & & \quad \vee (\exists y^{|N|}.\, z = \iota_2 y \wedge N^* y) \\
|M \supset N| & \stackrel{df}{=} & |M| \Rightarrow |N| & (M \supset N)^* z & \stackrel{df}{=} & \forall x^{|M|}.\, M^* x \supset N^*(zx) \\
|\forall x^\tau.M| & \stackrel{df}{=} & \tau \Rightarrow |M| & (\forall x^\tau.M)^* z & \stackrel{df}{=} & \forall x^\tau.\, M^*(zx) \\
|\exists x^\tau.M| & \stackrel{df}{=} & \tau \times |M| & (\exists x^\tau.M)^* z & \stackrel{df}{=} & (M\{\pi_1 z/x\})^*(\pi_2 z)
\end{array}$$

where the logical constants *true*, *false*, and the connective $\wedge$ in the definiens stand for their second-order encoding via $\forall$ and $\supset$ in $\mathcal{B}$ given in Section 3.1.1. Just as before the variable z in each case is assumed not to occur among the free variables in M or N. Additionally, in the clauses for $\supset$ and $\vee$ the variables x, y must be of the appropriate types and not be free in M or N.

Remark: In the definitions above $M^\bullet$ is a predicate of type $|M| \Rightarrow \Omega$. Alternatively, the clauses can be understood as defining a syntactic translation $M \mapsto M^\bullet[z]$ that turns M into a proposition with an extra free variable z. We will sometimes adopt this latter view in order to saves β-transformations.

Theorem 3.2.1 (Correctness of Extraction I)

- *For every well-formed formula M of $\mathcal{L}$, with free variables in context Δ, $|M|$ as defined above is a well-formed type and $M^\bullet$ a well-formed term of type $|M| \Rightarrow \Omega$, with the same free variables as M.*
- *Let M be a well-formed formula with a free variable x of type τ and t a well-formed term of type τ. Then, $|M\{t/x\}| = |M|$ and $M^\bullet\{t/x\} = (M\{t/x\})^\bullet$.*

Proof: The proof proceeds by induction on the structure of M. The interesting case is to verify $(\Diamond M)^\bullet\{t/x\} = (\Diamond M\{t/x\})^\bullet$. In this case we have to use that the action translation $\phi, c \mapsto \phi^c$ respects substitution:

$$\begin{aligned}
(\Diamond M)^\bullet\{t/x\} &= (\lambda z.(M^\bullet(\pi_2 z))^{\pi_1 z})\{t/x\} \\
&= \lambda z.(M^\bullet(\pi_2 z))^{\pi_1 z}\{t/x\} \\
&= \lambda z.(M^\bullet(\pi_2 z)\{t/x\})^{\pi_1 z\{t/x\}} \\
&= \lambda z.(M^\bullet\{t/x\}(\pi_2 z))^{\pi_1 z} \\
&= \lambda z.((M\{t/x\})^\bullet(\pi_2 z))^{\pi_1 z} \\
&= (\Diamond M\{t/x\})^\bullet
\end{aligned}$$

The third equation is due to the action preserving substitution, the fifth equation is the induction hypothesis. ∎

So much for the first part of the extraction process. The second part covering 3.4 and 3.5 proceeds by induction on the structure of derivations. To deal with the general case we wish to extract from every derivation of an $\mathcal{L}$-sequent

$$M_1, \ldots, M_k \vdash_\Delta M$$

and sequence $z_1, \ldots, z_k$ of arbitrary but fixed variables of types $|M_i|$, $i = 1, \ldots, k$, respectively, a well-formed constraint term

$$\Delta, z_1, \ldots, z_k \vdash t : |M| \tag{3.6}$$

and a derivation of the $\mathcal{B}$-sequent

$$M_1{}^* z_1, \ldots, M_k{}^* z_k \vdash_{\Delta, z_1, \ldots, z_k} M^* t. \tag{3.7}$$

In the case where both $k = 0$ and $\Delta = \emptyset$ this specializes to 3.4 and 3.5. We remark that in order to avoid renaming of variables we impose on the variables z_i the restriction that they be *fresh*, *i.e.* they must be different from all free variables that occur in the whole derivation tree of $M_1, \ldots, M_k \vdash_\Delta M$, which, because of rules $\forall I$, $\exists E$, *sub* may even be variables not contained in Δ. We will sometimes abbreviate the list $z_1, \ldots, z_k$ of variables by $\bar{z}$, the list $M_1, \ldots, M_k$ of hypotheses by Γ, and $M_1{}^* z_1, \ldots, M_k{}^* z_k$ by $\Gamma^* \bar{z}$.

There are several ways to define the envisaged translation. For instance we may use the fact that a derivation in $\mathcal{L}$, being nothing but a tree of rule applications, can be represented in linear notation by traversing the derivation tree in some fixed order (*e.g.* preorder) and noting down the names and other relevant information of the rules as they are visited. The translation could then be defined by induction along the structure of this 'term' of enriched rule names. What bits of data are actually required of each rule may be inferred by the following guideline: For each rule, given complete knowledge of its conclusion, the data must be sufficient to determine uniquely all premisses. To treat rule $\forall E$, for example, we need to record in addition to the rule name the term t that is substituted in order to work out the premisses $\Delta \vdash t : \tau$ and $\Delta, x \vdash M$ wff from the conclusion

$\forall x.\, M \vdash_\Delta M\{t/x\}$. So we might write $\forall E_t$ to record an application of rule $\forall E$. The most expensive rules in this respect, of course, are the 'cut' rules *cut* and *sub*. Anyway, if the guideline is obeyed for all rules, then we obtain a unique representation of the derivation of $M_1, \ldots, M_k \vdash_\Delta M$ in linear notation. We will use this approach in the next section to translate natural deduction proof trees for $\mathcal{L}$. They are much more economical than inference trees and also *cut* and *sub*-free.

At this point, however, we wish to define the extraction process without any intermediate steps and directly translate each application of an inference rule, and thus the tree as a whole. We are not reducing the amount of information necessary to represent derivations first and writing it down in linear fashion. The translation concerning (3.6) is shown in Figure 3-10 for the structural rules and in Figures 3-11, 3-12 for the logical rules. The other part of the translation concerning (3.7) we will not spell out explicitly since we are not interested in proofs within the base logic; it is contained in the proof of Lemma 3.2.2 stated below.

The first column in Figures 3-10, 3-11, 3-12 lists the name of each rule, the second shows the rule (scheme), and in the third the constraint terms along with their typings are given, by which the corresponding rule has to be replaced in the translation. How are these tables to be understood?

First, some technical remarks are in order. For simplicity we have left out all premisses that are judgements of well-formedness for formulae in the rules. These premisses do not contribute towards the construction of the constraint term. In contrast, premisses that are typings $\Delta \vdash t : \tau$ of some term t as in rules *sub* (see Fig. 3-10), or $\forall E$ and $\exists E$ (see Fig. 3-12) are necessary for typing the constraint term since here the term t enters the constraint term. In these cases, therefore, the complete derivation subtree that ends in premiss $\Delta \vdash t : \tau$ is taken over into the typing tree for the constraint term. Also, to understand the typing of the constraint term in these cases one has to bear in mind that $|M\{t/x\}| = |M|$.

rule		*derived typing of constraint term*
id	$M \vdash_\Delta M$	$\Delta, z^{\lvert M\rvert} \vdash z : \lvert M\rvert$
weak	$\dfrac{\Gamma, M \vdash_\Delta N}{\Gamma \vdash_\Delta N}$	$\dfrac{\Delta, \bar{z}, z^{\lvert M\rvert} \vdash t : \lvert N\rvert}{\Delta, \bar{z} \vdash t : \lvert N\rvert}$
weak	$\dfrac{\Gamma \vdash_{\Delta,x} M}{\Gamma \vdash_\Delta M}$	$\dfrac{\Delta, x, \bar{z} \vdash t : \lvert M\rvert}{\Delta, \bar{z} \vdash t : \lvert M\rvert}$
perm	$\dfrac{\Gamma \vdash_{\Delta,y,x,\Delta'} M}{\Gamma \vdash_{\Delta,x,y,\Delta'} M}$	$\dfrac{\Delta, y, x, \Delta', \bar{z} \vdash t : \lvert M\rvert}{\Delta, x, y, \Delta', \bar{z} \vdash t : \lvert M\rvert}$
perm	$\dfrac{\Gamma, N, M, \Gamma' \vdash_\Delta K}{\Gamma, M, N, \Gamma' \vdash_\Delta K}$	$\dfrac{\Delta, \bar{z}, v, u, \bar{z}' \vdash t : \lvert K\rvert}{\Delta, \bar{z}, u, v, \bar{z}' \vdash t : \lvert K\rvert}$
sub	$\dfrac{\Gamma\{t/x\} \vdash_\Delta M\{t/x\}}{\Gamma \vdash_{\Delta,x^\tau} M \quad \Delta \vdash t : \tau}$	$\dfrac{\Delta, \bar{z} \vdash s\{t/x\} : \lvert M\rvert}{\Delta, x^\tau, \bar{z} \vdash s : \lvert M\rvert \quad \Delta \vdash t : \tau}$
cut	$\dfrac{\Gamma \vdash_\Delta N}{\Gamma \vdash_\Delta M \quad \Gamma, M \vdash_\Delta N}$	$\dfrac{\Delta, \bar{z} \vdash s\{t/z\} : \lvert N\rvert}{\Delta, \bar{z} \vdash t : \lvert M\rvert \quad \Delta, \bar{z}, z^{\lvert M\rvert} \vdash s : \lvert N\rvert}\ z$ *fresh*

Figure 3–10: Constraint Extraction for Structural Rules of Lax Logic

The last remark concerns the cases of $\forall I$ and $\exists E$ (see Fig. 3–12): There the free variable x that gets bound by the rule application and the variable x in the constraint term that gets λ-abstracted in case $\forall I$ and substituted for in case $\exists E$ are the same. This is very convenient but not strictly necessary.

How does the translation work? Suppose we are given a derivation of

$$M_1, \ldots, M_k \vdash_\Delta M$$

and a sequence $z_1, \ldots, z_k$ of fresh variables of types $|M_i|$. The derivation tree is transformed according to the tables of Figures 3–10, 3–11, 3–12 into a tree of typings, by replacing each rule application by the corresponding typing rule for the constraint term. As mentioned before, every subtree that is a typing in the

rule		*derived typing of extracted term*
falseE	$\mathit{false} \vdash_\Delta M$	$\Delta, z^{\mathbf{0}} \vdash \Box z : \lvert M\rvert$
$\vee E$	$\dfrac{M \vee N \vdash_\Delta K}{M \vdash_\Delta K \quad N \vdash_\Delta K}$	$\dfrac{\Delta, z^{\lvert M\rvert+\lvert N\rvert} \vdash \mathit{case}_{x,y}(z,s,t) : \lvert K\rvert}{\Delta, x^{\lvert M\rvert} \vdash s : \lvert K\rvert \quad \Delta, y^{\lvert N\rvert} \vdash t : \lvert K\rvert}\ x, y\ \mathit{fresh}$
$\vee I_l$	$M \vdash_\Delta M \vee N$	$\Delta, z^{\lvert M\rvert} \vdash \iota_1 z : \lvert M\rvert + \lvert N\rvert$
$\vee I_r$	$N \vdash_\Delta M \vee N$	$\Delta, z^{\lvert N\rvert} \vdash \iota_2 z : \lvert M\rvert + \lvert N\rvert$
trueI	$\vdash \mathit{true}$	$\vdash * : \mathbf{1}$
$\wedge I$	$\dfrac{\Gamma \vdash_\Delta M \wedge N}{\Gamma \vdash_\Delta M \quad \Gamma \vdash_\Delta N}$	$\dfrac{\Delta, \bar{z} \vdash (s,t) : \lvert M\rvert \times \lvert N\rvert}{\Delta, \bar{z} \vdash s : \lvert M\rvert \quad \Delta, \bar{z} \vdash t : \lvert N\rvert}$
$\wedge E_l$	$M \wedge N \vdash_\Delta M$	$\Delta, z^{\lvert M\rvert\times\lvert N\rvert} \vdash \pi_1 z : \lvert M\rvert$
$\wedge E_r$	$M \wedge N \vdash_\Delta N$	$\Delta, z^{\lvert M\rvert\times\lvert N\rvert} \vdash \pi_2 z : \lvert N\rvert$
$\supset I$	$\dfrac{\Gamma \vdash_\Delta M \supset N}{\Gamma, M \vdash_\Delta N}$	$\dfrac{\Delta, \bar{z} \vdash \lambda z.\, t : \lvert M\rvert \Rightarrow \lvert N\rvert}{\Delta, \bar{z}, z^{\lvert M\rvert} \vdash t : \lvert N\rvert}\ z\ \mathit{fresh}$
$\supset E$	$\dfrac{\Gamma, M \vdash_\Delta N}{\Gamma \vdash_\Delta M \supset N}$	$\dfrac{\Delta, \bar{z}, z^{\lvert M\rvert} \vdash tz : \lvert N\rvert}{\Delta, \bar{z} \vdash t : \lvert M\rvert \Rightarrow \lvert N\rvert}$

Figure 3–11: Constraint Extraction for Logical Rules of Lax Logic, I

original derivation is copied over to the typing tree of the constraint term. It can be seen that in this process the variables on the left side of $\vdash$ propagate down the tree while the constraint term on the right side propagates up the tree. Also, except for additional choices of variables that have to be made, *viz.* in rules *cut*, $\equiv$, $\supset I$, $\exists E$, $\Diamond F$, the translation is deterministic, and hence the constraint term uniquely determined. Finally, it is not difficult to convince oneself that every typing rule for the constraint term in Figures 3–10, 3–11, 3–12 is a *derived* rule, so that in fact we end up with a valid derivation of a typing $\Delta, z_1, \ldots, z_k \vdash t : \lvert M\rvert$.

The correctness of the second part of our constraint extraction process is summarized in the following theorem:

Theorem 3.2.2 (Correctness of Extraction II)

- *Given a derivation of* $M_1, \ldots, M_k \vdash_\Delta M$ *and a sequence* $z_1, \ldots, z_k$ *of fresh*

variables of types $|M_i|$, $i = 1, \ldots, k$, respectively. Then, the translation of this derivation yields a well-formed term t of type $|M|$ with free variables among $\Delta, z_1, \ldots, z_k$.

- *For this term there is a valid derivation of the sequent*

$$M_1^* z_1, \ldots, M_k^* z_k \vdash_{\Delta, z_1, \ldots, z_k} M^* t.$$

Proof: The first part of the theorem has been covered by the remarks above. As to the second part we give a direct argument only for $\Diamond F$. The other rules can be treated in a similar way. The proofs are all straightforward and make use of the properties of a notion of constraint and the equation $M^*\{t/x\} = (M\{t/x\})^*$. For the rules *NatInd* and *ListInd* one needs induction in $\mathcal{B}$. It is remarked that for the closed propositional fragment the second part follows from the category theoretic interpretation of Section 5.2 (*cf.* Corollary 5.2.24).

We have to find a derivation for

$$\Gamma^* \bar{z}, (\Diamond M)^* z \vdash_{\Delta, \bar{z}, z} (\Diamond N)^*(\pi_1 z, t\{\pi_2 z/w\})$$

under the inductional assumption that we are given a derivation

$$\Gamma^* \bar{z}, M^* w \vdash_{\Delta, \bar{z}, w} N^* t.$$

Now, by definition $(\Diamond M)^* z$ is the proposition $(M^*(\pi_2 z))^{\pi_1 z}$ and the assertion $(\Diamond N)^*(\pi_1 z, t\{\pi_2 z/w\})$ is the proposition $(N^*(\pi_2(\pi_1 z, t\{\pi_2 z/w\})))^{\pi_1(\pi_1 z, t\{\pi_2 z/w\})}$. The rules of $\mathcal{B}$ allow us to prove

$$\vdash_{\Delta, \bar{z}, z} (N^*(\pi_2(\pi_1 z, t\{\pi_2 z/w\})))^{\pi_1(\pi_1 z, t\{\pi_2 z/w\})} = (N^*(t\{\pi_2 z/w\}))^{\pi_1 z},$$

so that the goal can be reduced to the equivalent

$$\Gamma^* \bar{z}, (M^*(\pi_2 z))^{\pi_1 z} \vdash_{\Delta, \bar{z}, z} (N^*(t\{\pi_2 z/w\}))^{\pi_1 z}.$$

But this can be obtained from the inductional assumption by invoking rule ***sub*** substituting $\pi_2 z$ for w, and by the properties of a notion of constraint. ∎

	rule	*derived typing of extracted term*
$\forall I$	$\dfrac{\Gamma \vdash_\Delta \forall x.\, M}{\Gamma \vdash_{\Delta, x^\tau} M}$	$\dfrac{\Delta, \bar{z} \vdash \lambda x.\, t : \tau \Rightarrow \lvert M\rvert}{\Delta, x^\tau, \bar{z} \vdash t : \lvert M\rvert}$
$\forall E$	$\dfrac{\forall x.\, M \vdash_\Delta M\{t/x\}}{\Delta \vdash t : \tau}$	$\dfrac{\Delta, z^{\tau \Rightarrow \lvert M\rvert} \vdash zt : \lvert M\rvert}{\Delta \vdash t : \tau}$
$\exists I$	$\dfrac{M\{t/x\} \vdash_\Delta \exists x.\, M}{\Delta \vdash t : \tau}$	$\dfrac{\Delta, z^{\lvert M\rvert} \vdash (t, z) : \tau \times \lvert M\rvert}{\Delta \vdash t : \tau}$
$\exists E$	$\dfrac{\exists x.\, M \vdash_\Delta N}{M \vdash_{\Delta, x^\tau} N}$	$\dfrac{\Delta, z^{\tau \times \lvert M\rvert} \vdash t\{\pi_1 z/x\}\{\pi_2 z/y\} : \lvert N\rvert}{\Delta, x^\tau, y^{\lvert M\rvert} \vdash t : \lvert N\rvert}\, y$ *fresh*
$\Diamond I$	$M \vdash_\Delta \Diamond M$	$\Delta, z^{\lvert M\rvert} \vdash ([\,], z) : \Omega^* \times \lvert M\rvert$
$\Diamond M$	$\Diamond\Diamond M \vdash_\Delta \Diamond M$	$\Delta, z^{\Omega^* \times (\Omega^* \times \lvert M\rvert)} \vdash (\pi_1 \pi_2 z \mathbin{@} \pi_1 z, \pi_2 \pi_2 z) : \Omega^* \times \lvert M\rvert$
$\Diamond F$	$\dfrac{\Gamma, \Diamond M \vdash_\Delta \Diamond N}{\Gamma, M \vdash_\Delta N}$	$\dfrac{\Delta, \bar{z}, z^{\Omega^* \times \lvert M\rvert} \vdash (\pi_1 z, t\{\pi_2 z/w\}) : \Omega^* \times \lvert N\rvert}{\Delta, \bar{z}, w^{\lvert M\rvert} \vdash t : \lvert N\rvert}\, w$ *fresh*
ι	$\dfrac{\iota\phi_1, \ldots, \iota\phi_k \vdash_\Delta \iota\phi}{\phi_1, \ldots, \phi_k \vdash_\Delta \phi}$	$\Delta, z_1^1, \ldots, z_k^1 \vdash * : 1$
NatInd	$\dfrac{\Gamma \vdash_\Delta \forall y.\, M}{\Gamma \vdash_\Delta M\{0/y\} \quad \Gamma, M \vdash_{\Delta, y} M\{succ\, y/y\}}$	$\dfrac{\Delta, \bar{z} \vdash natrec(a, \lambda y.\, \lambda w.\, t) : \mathbf{N} \Rightarrow \lvert M\rvert}{\Delta, \bar{z} \vdash a : \lvert M\rvert \quad \Delta, y, \bar{z}, w^{\lvert M\rvert} \vdash t : \lvert M\rvert}$
ListInd	$\dfrac{\Gamma \vdash_\Delta \forall y.\, M}{\Gamma \vdash_\Delta M\{[\,]/y\} \quad \Gamma, M \vdash_{\Delta, x, y} M\{x :: y/y\}}$	$\dfrac{\Delta, \bar{z} \vdash listrec(a, \lambda x.\, \lambda y.\, \lambda w.\, t) : \tau^* \Rightarrow \lvert M\rvert}{\Delta, \bar{z} \vdash a : \lvert M\rvert \quad \Delta, x, y, \bar{z}, w^{\lvert M\rvert} \vdash t : \lvert M\rvert}$

Figure 3–12: Constraint Extraction for Logical Rules of Lax Logic, II

The extraction process has been hard-wired for the particular notion of constraint $(\Omega^*, [\,], @)$. For an arbitrary notion of constraint $(\mathbf{C}, 1, \cdot)$ one simply replaces the definition of the constraint type of formulae $\Diamond M$ by

$$|\Diamond M| \stackrel{df}{=} \mathbf{C} \times |M|$$

and the translation of the rules $\Diamond I, \Diamond M, \Diamond F$ in Figure 3–12 as follows

$M \vdash_\Delta \Diamond M$	$\Delta, z^{\lvert M\rvert} \vdash (1, z) : \mathbf{C} \times \lvert M\rvert$
$\Diamond\Diamond M \vdash_\Delta \Diamond M$	$\Delta, z^{\mathbf{C}\times(\mathbf{C}\times\lvert M\rvert)} \vdash (\pi_1\pi_2 z \cdot \pi_1 z, \pi_2\pi_2 z) : \mathbf{C} \times \lvert M\rvert$
$\dfrac{\Gamma, \Diamond M \vdash_\Delta \Diamond N}{\Gamma, M \vdash_\Delta N}$	$\dfrac{\Delta, \bar{z}, z^{\mathbf{C}\times\lvert M\rvert} \vdash (\pi_1 z, t\{\pi_2 z/w\}) : \mathbf{C} \times \lvert N\rvert}{\Delta, \bar{z}, w^{\lvert M\rvert} \vdash t : \lvert N\rvert}\, w \textit{ fresh}$

The correctness theorems 3.2.1, 3.2.2 then carry over to the general case.

$$\dfrac{\dfrac{\Gamma, \Diamond M \vdash_\Delta \Diamond N}{\Gamma, \Diamond M \vdash_\Delta \Diamond\Diamond N}\,\Diamond F \quad \dfrac{\Gamma, \Diamond M, \Diamond\Diamond N \vdash_\Delta \Diamond N}{\dfrac{\Diamond\Diamond N, \Gamma, \Diamond M \vdash_\Delta \Diamond N}{\Diamond\Diamond N \vdash_\Delta \Diamond N}\,\textit{weak}\cdots\textit{weak}}\,\Diamond M}{\Gamma, M \vdash_\Delta \Diamond N}\,\textit{cut}\ \textit{perm}\cdots\textit{perm}$$

$$\dfrac{\dfrac{\Delta, \bar{z}, z \vdash (\pi_1\pi_2(\pi_1 z, t\{\pi_2 z/w\}) @ \pi_1(\pi_1 z, t\{\pi_2 z/w\}),\ \pi_2\pi_2(\pi_1 z, t\{\pi_2 z/w\}))}{\dfrac{\Delta, \bar{z}, z \vdash (\pi_1 z, t\{\pi_2 z/w\})}{\Delta, \bar{z}, w \vdash t} \quad \dfrac{\Delta, \bar{z}, z, y \vdash (\pi_1\pi_2 y @ \pi_1 y,\ \pi_2\pi_2 y)}{\dfrac{\Delta, y, \bar{z}, z \vdash (\pi_1\pi_2 y @ \pi_1 y,\ \pi_2\pi_2 y)}{\Delta, y \vdash (\pi_1\pi_2 y @ \pi_1 y,\ \pi_2\pi_2 y)}}}$$

Figure 3–13: Derivation of $\Diamond L$ from $\Diamond F, \Diamond M$ and Extracted Constraint Term

Now let us look at a simple example. In Figure 3–13 the upper part depicts a derivation of rule $\Diamond L$ from rules $\Diamond F$ and $\Diamond M$, rule *cut*, and a number of applications of the structural rules *perm*, *weak*. In the lower part the corresponding constraint term is constructed. The types of the terms and variables have been omitted to reduce the size of the tree. Given that

$$\begin{array}{lll} w & \text{has type} & |M| \\ z & \text{has type} & |\Diamond M| = \Omega^* \times |M| \\ y & \text{has type} & |\Diamond\Diamond N| = \Omega^* \times (\Omega^* \times |N|) \end{array}$$

one may convince oneself that the types can be filled in so that every step in the construction is a valid (derived) typing, and that the resulting constraint term

$$(\pi_1\pi_2(\pi_1 z, t\{\pi_2 z/w\}) \mathbin{@} \pi_1(\pi_1 z, t\{\pi_2 z/w\}\ ,\ \pi_2\pi_2(\pi_1 z, t\{\pi_2 z/w\}))$$

is well-formed of type $|\Diamond N| = \Omega^* \times |N|$ with free variables in $\Delta, \bar{z}, z$ whenever t is well-formed of type $|\Diamond N|$ with free variables in $\Delta, \bar{z}, w$. We can now bring into play the equality axioms for projections and pairing to simplify the constraint term somewhat, *viz.* we can prove

$$\begin{aligned} \vdash_{\Delta,\bar{z},z} \quad & (\pi_1\pi_2(\pi_1 z, t\{\pi_2 z/w\}) \mathbin{@} \pi_1(\pi_1 z, t\{\pi_2 z/w\}\ ,\ \pi_2\pi_2(\pi_1 z, t\{\pi_2 z/w\})) \\ & = (\pi_1 t\{\pi_2 z/w\} \mathbin{@} \pi_1 z\ ,\ \pi_2 t\{\pi_2 z/w\}). \end{aligned}$$

We may sum up all this by saying that the translation of rule

$$\frac{\Gamma, \Diamond M \vdash_\Delta \Diamond N}{\Gamma, M \vdash_\Delta \Diamond N}\Diamond L$$

for extracting the constraint term is given by

$$\frac{\Delta, \bar{z}, z^{\Omega^* \times |M|} \vdash (\pi_1 t\{\pi_2 z/w\} \mathbin{@} \pi_1 z\ ,\ \pi_2 t\{\pi_2 z/w\}) : \Omega^* \times |N|}{\Delta, \bar{z}, w^{|M|} \vdash t : \Omega^* \times |N|} w \textit{ fresh.}$$

We may add this translation for $\Diamond L$ to Figure 3–12 and preserve the second half of Theorem 3.2.2 since it holds that from every derivation of

$$\Gamma^\# \bar{z}, M^\# w \vdash_{\Delta,\bar{z},w} \Diamond N^\# t$$

can be obtained a derivation of

$$\Gamma^\# \bar{z}, (\Diamond M)^\# z \vdash_{\Delta,\bar{z},z} (\Diamond N)^\#(\pi_1 t\{\pi_2 z/w\} \mathbin{@} \pi_1 z\ ,\ \pi_2 t\{\pi_2 z/w\})$$

where the type information is now suppressed. This follows from the proof of Theorem 3.2.2 and the construction of the constraint term.

Another important derived rule for which we need constraint extraction is the equality substitution rule

$$\frac{\Gamma \vdash_\Delta M\{s/x\}}{\Gamma \vdash_\Delta \iota(s =_\tau t) \quad \Gamma \vdash_\Delta M\{t/x\}}\ \mathit{Subst}$$

The algorithm for deriving this rule for all instances of formula M is implicit in the proof of lemma 3.1.14. If this construction was made explicit and the constraint terms evaluated as above for $\Diamond L$, then one would find that modulo provable equality of constraint terms all the cases can be subsumed by the following typing rule:

$$\frac{\Delta, \bar{z} \vdash b : |M|}{\Delta, \bar{z} \vdash a : 1 \quad \Delta, \bar{z} \vdash b : |M|}$$

Thus, substitution of equals for equals in a formula does not change the constraint term. This typing rule henceforth shall be regarded as the translation of *Subst* into constraint terms.

3.2.1 Application of Constraint Extraction

Let us now discuss some examples for the kind of use that we are going to make of constraint extraction, based on the particular notion of constraint $(\Omega^*, [\,], \mathbf{Q})$. This section is a summary of the technical issues involved in the application examples of Chapter 4.

For the purpose of this section we will focus on universal quantification $\forall$ as the major connector to formulate specifications. A behavioural specification in the base logic of a piece of hardware, so let us assume, is a proposition of the form $\forall y^\beta.\psi$. The variable y might range over the possible values of an input signal, β being the type of signal values, and ψ might express some property of the output produced by the piece of hardware in response to the input signal. For simplicity assume we are dealing with single input circuits.

Suppose, as in Chapter 2, we wanted to organize formal verification in a top-down fashion, *i.e.* to break down the initial specification into sub-specifications

and verify that provided we can find correct implementations for all the sub-specifications composing the implementations in some suitable way results in a circuit that satisfies the initial specification. The point is that this step can already be verified *before* the sub-specifications are implemented. Formally, this verification step, or *refinement* as it is sometimes called, amounts to proving a proposition of the form

$$(\forall x_1.\, \phi_1 \wedge \cdots \wedge \forall x_n.\, \phi_n) \supset \forall y.\, \psi \tag{3.8}$$

where n counts the number of subcomponents. For example, if the factorial function is to be built from an incrementor subcomponent, the refinement would read

$$\forall x_1.\, inc(x_1) = succ(x_1) \quad \supset \quad \forall y.\, fac(y+1) = (y+1) \cdot fac(y)$$

Given (3.8) is proven the remaining task is to find implementations satisfying the sub-specifications $\forall x_i.\, \phi_i$. But because these sub-specifications have been formulated before it is decided how they should be implemented, in particular, they cannot be expected to allow for all possible input constraints. In fact, it appears more realistic to assume, as we will do here, that specifications do not take into account input constraints at all. In other words, it will happen that the implementations do not exactly satisfy the sub-specifications, but instead

$$\forall x_i.\, \gamma_i \supset \phi_i$$

where the γ_i, $i = 1, \ldots, n$ are certain sufficiently strong input constraints. If this is the case we are in bad shape with our top-down verification since then all the previous refinement steps must be verified again: the fact that (3.8) is true has become worthless. For instance, if the incrementor is implemented over bit-vectors, it will come with an overflow constraint $\gamma_1 \equiv x_1 < 2^k$ to satisfy

$$\forall x_1.\, \gamma_1 \supset inc(x_1) = succ(x_1).$$

The problem can be avoided by formulating the refinement statement in lax

logic, more specifically replacing (3.8) by

$$(\forall x_1.\, \Diamond \iota \phi_1 \wedge \cdots \wedge \forall x_n.\, \Diamond \iota \phi_n) \supset \forall y.\, \Diamond \iota \psi$$

where the modal $\Diamond$ is put in to anticipate the input constraints γ_i. To simplify the situation even more consider a refinement goal

$$M \;\equiv\; (\forall x^{\alpha}.\, \Diamond \iota \phi) \supset \forall y^{\beta}.\, \Diamond \iota \psi$$

for a single subcomponent, where ϕ and ψ are well-formed propositions such that x^{α} is the only free variable of ϕ and y^{β} is the only free variable of ψ. Modifying the example above, for instance, yields such a formula, *viz.*

$$\forall x.\, \Diamond(inc(x) = succ(x)) \;\supset\; \forall y.\, \Diamond(fac(y+1) = (y+1) \cdot fac(y)).$$

From now on, let us look at the general case. We will see that from a proof of M, *i.e.* a derivation of

$$\vdash (\forall x^{\alpha}.\, \Diamond \iota \phi) \supset \forall y^{\beta}.\, \Diamond \iota \psi \tag{3.9}$$

we can extract a constraint transforming function F such that if $\forall x.\, \phi$ holds under some input constraint γ then $\forall y.\, \psi$ holds under input constraint $F\gamma$, *i.e.*

$$\vdash (\forall x^{\alpha}.\, \gamma \supset \phi) \supset \forall y^{\beta}.\, (F\gamma) \supset \psi \tag{3.10}$$

is derivable. Then no matter what input constraint γ has to be introduced eventually in sub-specification $\forall x.\, \phi$ in order to make it implementable the refinement verification does not need to be redone. The refinement (3.10) as the translated version of (3.9) works for all γ; it guarantees that there is always a constraint $F\gamma$ for the specification $\forall y.\, \psi$ of the composite circuit.

Here is how (3.10) can be obtained through constraint extraction: The constraint type of M can readily be computed from the definition given in the previous Section 3.2:

$$|M| \;\equiv\; |(\forall x^{\alpha}.\, \Diamond \iota \phi) \supset \forall y^{\beta}.\, \Diamond \iota \psi|$$

$$\begin{aligned}
&\equiv |\forall x^{\alpha}.\Diamond\iota\phi| \Rightarrow |\forall y^{\beta}.\Diamond\iota\psi| \\
&\equiv (\alpha \Rightarrow |\Diamond\iota\phi|) \Rightarrow (\beta \Rightarrow |\Diamond\iota\psi|) \\
&\equiv (\alpha \Rightarrow \Omega^{*} \times |\iota\phi|) \Rightarrow (\beta \Rightarrow \Omega^{*} \times |\iota\psi|) \\
&\equiv (\alpha \Rightarrow \Omega^{*} \times \mathbf{1}) \Rightarrow (\beta \Rightarrow \Omega^{*} \times \mathbf{1})
\end{aligned}$$

As to the constraint predicate $M^{\#}$ one computes

$$\begin{aligned}
\vdash\ M^{\#}t &\equiv ((\forall x^{\alpha}.\Diamond\iota\phi) \supset \forall y^{\beta}.\Diamond\iota\psi)^{\#}t \\
&\equiv \forall z^{\alpha\Rightarrow\Omega^{*}\times\mathbf{1}}.\,(\forall x^{\alpha}.\Diamond\iota\phi)^{\#}z \supset (\forall y^{\beta}.\Diamond\iota\psi)^{\#}(tz) \\
&\equiv \forall z^{\alpha\Rightarrow\Omega^{*}\times\mathbf{1}}.\,(\forall x^{\alpha}.(\Diamond\iota\phi)^{\#}(zx)) \supset \forall y^{\beta}.(\Diamond\iota\psi)^{\#}(tzy) \\
&\equiv \forall z^{\alpha\Rightarrow\Omega^{*}\times\mathbf{1}}.\,(\forall x^{\alpha}.((\iota\phi)^{\#}\pi_2(zx))^{\pi_1(zx)}) \supset \forall y^{\beta}.((\iota\psi)^{\#}(\pi_2(tzy)))^{\pi_1(tzy)} \\
&\equiv \forall z^{\alpha\Rightarrow\Omega^{*}\times\mathbf{1}}.\,(\forall x^{\alpha}.\phi^{\pi_1(zx)}) \supset \forall y^{\beta}.\psi^{\pi_1(tzy)}
\end{aligned}$$

This means by Lemma 3.2.2 that every derivation of (3.9) induces a constraint term

$$t : (\alpha \Rightarrow \Omega^{*} \times \mathbf{1}) \Rightarrow (\beta \Rightarrow \Omega^{*} \times \mathbf{1})$$

and a proof of $\vdash M^{\#}t$, *i.e.* a derivation of

$$\vdash \forall z^{\alpha\Rightarrow\Omega^{*}\times\mathbf{1}}.\,(\forall x^{\alpha}.\phi^{\pi_1(zx)}) \supset \forall y^{\beta}.\psi^{\pi_1(tzy)}.$$

This is a universally quantified proposition and we may specialize z to any well-formed term of type $\alpha \Rightarrow \Omega^{*} \times \mathbf{1}$. For instance, we can let z be the term $\lambda x^{\alpha}.([\gamma], *)$, where γ is any proposition with x as free variable. With a little equality reasoning we obtain a derivation of

$$\vdash (\forall x^{\alpha}.\gamma \supset \phi) \supset \forall y^{\beta}.\psi^{\pi_1(t(\lambda x^{\alpha}.([\gamma],*))y)} \tag{3.11}$$

This proposition has almost the required form (3.10) except that the constraint $\pi_1(t(\lambda x^{\alpha}.([\gamma], *))y)$ in general is a list of more than one proposition. But this can be repaired using the natural map $\sqcap : \Omega^{*} \Rightarrow \Omega$ with the property $\psi^{c} \cong (\sqcap c) \supset \psi$. This map was defined in Section 3.1.2. If we put $F\gamma$ to be

$$F\gamma \stackrel{\mathrm{df}}{=} \sqcap(\pi_1(t(\lambda x^{\alpha}.([\gamma], *))y))$$

then it is not difficult to prove (3.10) from (3.11). This completes the discussion of this rather abstract example of constraint extraction. More concrete examples will be discussed later in Chapter 4 (*e.g.* in Section 4.1.4).

Another way of using constraint extraction that will be important for our purposes can be explained by considering (3.10). Since γ in (3.10) is completely arbitrary, we may choose $\gamma = \phi$ in particular, so that (3.10) becomes

$$\vdash (\forall x^{\alpha}.\, \phi \supset \phi) \supset (\forall y^{\beta}.\, (F\phi) \supset \psi)$$

where $F\phi = \Pi(\pi_1(t(\lambda x^{\alpha}.\, ([\phi], *))y))$. But now the antecedent of the outermost $\supset$ has become trivially provable, whence this is equivalent to

$$\vdash \forall y^{\beta}.\, (F\phi) \supset \psi. \tag{3.12}$$

Compare this with the original goal (3.9): formally, we have turned a derivation of

$$\vdash \Theta \supset \forall y^{\beta}.\, \Diamond \iota\psi \quad \text{where} \quad \Theta \stackrel{df}{\equiv} \forall x^{\alpha}.\, \Diamond \iota\phi \tag{3.13}$$

into a derivation of

$$\vdash \forall y^{\beta}.\, \theta \supset \psi \quad \text{where} \quad \theta \stackrel{df}{\equiv} F\phi. \tag{3.14}$$

Besides translating a proof in $\mathcal{L}$ into a proof in $\mathcal{B}$, we have managed to get rid of the antecedent Θ in exchange for replacing the modal $\Diamond$ in the succedent $\forall y^{\beta}.\, \Diamond \iota\psi$ by the constraint θ. If we ignore the modal $\Diamond$ and the embedding operator ι for a moment, then what happens is that $\Theta \supset \forall y.\, \psi$ is replaced by $\forall y.\, \theta \supset \psi$. Of course, $\Theta \supset \forall y.\, \psi$ is always equivalent to $\forall y.\, \Theta \supset \psi$ by the property of predicate logic[5]. However, the important observation here is that θ has y as a free variable and may thus depend on variable y while Θ, coming from outside of the scope of the quantification, *cannot* depend on y. In other words we have transformed the global assumption Θ into a (local) assumption θ that depends on the local variable y.

[5] *up to renaming of the bound variable* y

What is the intuition behind this? $\Theta \equiv \forall x.\, \Diamond\iota\phi$ is a generalization of $\Diamond\iota\phi$ over all x of which, in a proof of the implication $\Theta \supset \forall y^{\beta}.\, \Diamond\iota\psi$, only a certain number of special instances $\Diamond\iota\phi\{a/x\}$ will actually be referred to in general. All other cases are not strictly necessary in the proof. In the extreme case the proof might not depend on any instance, whence the assumption Θ would not be necessary at all. This information basically is recorded in the extracted constraint term t which is the main constituent of θ. Different derivations of (3.13) will produce different constraint terms t, and hence different local assumptions θ.

Let us demonstrate for a simple case that in the passage from (3.13) to (3.14), θ in fact reproduces the special instances of $\Diamond\iota\psi$ that are used in the derivation (3.13). In order to get hold of specific constraint terms t we must further specialize the example. Suppose, proposition ψ is

$$\psi \stackrel{df}{\equiv} \phi\{f_l/x\} \vee \phi\{f_r/x\}$$

where f_r and f_l are two well-formed terms of type α with y^{β} being their only free variable. For the following abbreviate $\phi\{f_l/x\}$ by $\phi\{f_l\}$, and similarly $\phi\{f_r/x\}$ by $\phi\{f_r\}$. For this particular choice of ψ we can give two different proofs of (3.13). If we ignore the $\Diamond$ and ι operators, then the goal is to prove

$$(\forall x.\, \phi) \supset \forall y.\, \phi\{f_l\} \vee \phi\{f_r\}$$

The succedent (of $\supset$) claims that ϕ holds of at least one of two particular instances while the antecedent postulates that it is true of all instances. Depending on which of the cases in the succedent is picked we get two different proofs of the goal. These proofs, with $\Diamond$s and ιs put back in, are depicted as a single derivation tree in the upper part of Figure 3–14.

Both derivations are identical except for the subtrees starting with rule *cut*. The subtrees are parameterized in the index $i \in \{l, r\}$ of f_i. For $i = l$ the derivation in the right subtree proves the disjunction $\phi\{f_l\} \vee \phi\{f_r\}$ from its left disjunct $\phi\{f_l\}$ and for $i = r$ from its right disjunct $\phi\{f_r\}$. In both cases the disjunct is inferred from the global assumption $\forall x.\, \Diamond\iota\phi$ by specializing it

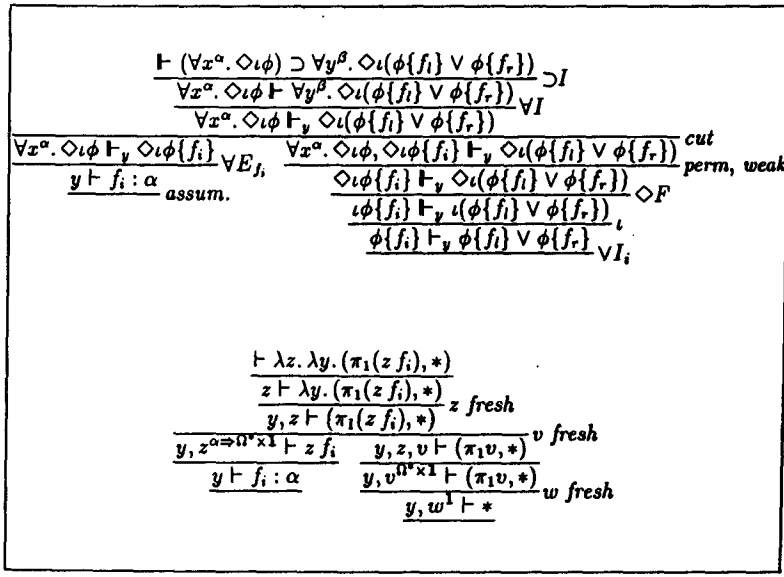


Figure 3–14: A Simple Application of Constraint Extraction

appropriately. This is done in the left subtree. Thus, this example exhibits a very simple case of two proofs of (3.13) which differ in the instances of the global assumption $\Theta \equiv \forall x.\, \Diamond\iota\phi$ they make use of.

The lower part of Figure 3–14 shows the translation for extracting the corresponding constraint terms. Again it is parameterized in $i \in \{l, r\}$ and also all typing information is omitted. We find that we get the constraint terms

$$t_i \equiv \lambda z.\, \lambda y.\, (\pi_1(z\, f_i), *) \qquad i \in \{l, r\}$$

both well-formed of type $(\alpha \Rightarrow \Omega^* \times 1) \Rightarrow (\beta \Rightarrow \Omega^* \times 1)$ where z is a fresh variable of type $\alpha \Rightarrow \Omega^* \times 1$. For these terms the local assumption θ of (3.14) specializes to

$$\theta_i = F_i\phi = \sqcap(\pi_1((\lambda z.\, \lambda y.\, (\pi_1(z\, f_i), *))(\lambda x^{\alpha}.\, ([\phi], *))y))$$

which in turn can be simplified as follows

$$\vdash_y \quad \theta_i = \sqcap(\pi_1((\lambda z.\, \lambda y.\, (\pi_1(z\, f_i), *))(\lambda x^{\alpha}.\, ([\phi], *))y))$$

$$
\begin{aligned}
&= \Pi(\pi_1(\lambda y.(\pi_1((\lambda x^{\alpha}.([\phi],*))\, f_i),*)\, y)) \\
&= \Pi(\pi_1(\pi_1((\lambda x^{\alpha}.([\phi],*))\, f_i),*)) \\
&= \Pi(\pi_1(\pi_1([\phi\{f_i\}],*),*)) \\
&= \Pi[\phi\{f_i\}] \\
&= \phi\{f_i\} \wedge \mathit{true} \\
&\cong \phi\{f_i\}
\end{aligned}
$$

using equality axioms for product, function, and list types, and the properties of Π. To sum up we find that the two $\mathcal{L}$-derivations of

$$\vdash (\forall x^{\alpha}.\Diamond_{\iota}\phi) \supset \forall y^{\beta}.\Diamond_{\iota}(\phi\{f_l\} \vee \phi\{f_r\})$$

are transformed into a $\mathcal{B}$-derivation of

$$\vdash \forall y^{\beta}.\phi\{f_i\} \supset (\phi\{f_l\} \vee \phi\{f_r\}) \qquad i \in \{l,r\}.$$

It can be seen that the local constraints θ_i are in fact different for the two derivations and that they reproduce that particular instance of the global assumption that is used in the proof. This confirms that the local assumptions θ_i, as promised, precisely capture to what extent the proof depends on the global assumption $\forall x^{\alpha}.\Diamond_{\iota}\phi$.

What might this be good for? It provides a mechanism for discharging any global hypothesis of form $\forall x.\Diamond_{\iota}\phi$ by translating a proof from $\mathcal{L}$ into $\mathcal{B}$. It recovers from the hypothesis only those bits that were actually used in the proof and puts them back into the local context. This mechanism is in its most general form when applied to the particular hypothesis[6]

$$\textit{let's-not-bother} \stackrel{df}{\cong} \forall z^{\Omega}.\Diamond_{\iota z}$$

In the course of a proof that is done in the context of this hypothesis we obviously can solve any subgoal $\Diamond_{\iota}\gamma$ no matter what γ is. The result may be viewed

[6]The name of this formula was suggested by Rod Burstall

as an *incomplete* proof in which at certain points 'holes' have been left by referring to *let's-not-bother*. Whenever we decide to do so the global hypothesis *let's-not-bother* can be discharged by constraint extraction as explained, replacing all $\Diamond$s within the formula proven by certain local assumptions. It can be expected from the discussion above that this roughly results in all the individual subgoals, which have been left 'unsolved' using *let's-not-bother*, popping up in their appropriate contexts. The crucial property of *let's-not-bother* making this work is that is has a distinguished constraint term

$$? \stackrel{df}{\equiv} \lambda z.([z], *)$$

of type $|\textit{let's-not-bother}| \equiv \Omega \Rightarrow (\Omega^* \times 1)$ that 'solves' it in the sense that

$$\textit{let's-not-bother}^\# ? \cong \textit{true}$$

becomes trivially provable.

There are several situations when it is convenient to consider such incomplete proofs (or rather: complete proof under hypothesis *let's-not-bother*) adequate first approximations of a proper proof of some theorem, and which therefore should be available for use in other derivations. Firstly, at the one extreme the unsolved subgoal may be inconsistent. This does not preclude, however, that the incomplete proof constructed has done *some* useful job. Often parts of the proof do not depend on this inconsistent subgoal at all or merely on a specialization of it which may well be consistent. Secondly, the subgoal is consistent but is not provable (or at least no proof is known) in the present context. This leaves the possibility that it may later become provable when this incomplete proof is used in a different context. Finally, at the other extreme the assumption may be provable but simply is considered less important at the present stage and consequently its proof, distracting from the main objective, better is delayed until later. In Chapter 4 it will be demonstrated that each of these cases does indeed arise in practical examples of hardware verification (*e.g.* in Sections 4.1.1, 4.1.3).

A concluding note: Although the distinguished formula *let's-not-bother* has the potential to prove, within $\mathcal{L}$, any proposition of $\mathcal{B}$, it does not produce a contradiction. We recall that the hypothesis $\forall z^{\Omega}.\, \Diamond \iota z$ is consistent in $\mathcal{L}$ (*cf.* Theorem 3.1.20), while of course $(\forall z^{\Omega}.\, z) \equiv \mathit{false}$ in $\mathcal{B}$ is inconsistent.

3.3 Natural Deduction Proofs and Constraint Extraction

We have chosen to present lax logic in sequent style since we feel that this is the most adequate and concise way for formally defining predicate logics. However, for the presentation of concrete application examples it will be more convenient to construct the relevant proofs in natural deduction rather than sequent style. On the one hand this yields much more compact proofs which is important if they are to fit on a single page or less. On the other hand the process of constraint extraction can be explained as a simple mapping from natural deduction proof trees into λ-terms, in fact, the proof tree itself is the extracted constraint term, except that some information is thrown away.

It will be assumed that the reader is familiar with natural deduction proofs in ordinary predicate logic (*e.g.* [Pra65]) and with the relationship between natural deduction and sequent style presentations. The non-standard aspect of lax logic is the idea of distinguishing between a base logic and a first-order extension thereof with embedding operator ι, and, of course, the modal one-place connective $\Diamond$. So, we will pay special attention to these aspects only.

In Figure 3–15 the logical rules of lax logic are depicted as natural deduction rules. Note, the notion of a context of hypothesis and variables is implicit in natural deduction, so that the structural rules have no counterparts. They are automatically built in by the fact that hypotheses and variables are ubiquitous.

The usual side conditions apply to rules $\forall I$, $\exists E$, *NatInd*, and *ListInd*. These are not mentioned in Figure 3–15. The only other rule with a side condition is ι,

the two instances of which that we are going to use in the examples are

$$\frac{\iota\psi}{\iota\phi}\,\iota \qquad \frac{\iota\psi}{\iota\psi}\,\iota$$

These rules correspond to the two instances of the inference rule

$$\frac{\iota\phi_1, \ldots, \iota\phi_k \vdash_\Delta \iota\psi}{\phi_1, \ldots, \phi_k \vdash_\Delta \psi}\,\iota$$

with number of hypotheses $k = 0$ and $k = 1$. In applying these rules it is understood that ϕ and ψ are propositions in the base logic and that $\phi_1, \ldots, \phi_k \vdash_\Delta \psi$ is actually derivable in the base logic, and that this derivation is given constructively.

Since derivations in the base logic, or proofs of formulae of the shape $\iota\phi$ in lax logic, do not contribute to constraint information — the extracted relevant constraint terms always evaluate to the trivial element $*$ of type 1 — in all of the examples no derivation in the base logic will ever be formally written out in detail. Instead we will appeal to the imagination of the reader and simply treat such proofs as implicit side conditions in every application of rule ι. Notice, in particular, that all equality reasoning is among these proofs.

Another aspect, which would be part of a fully formal proof but will be left out in the examples, are the typing judgements. For instance, a second premiss of rules $\forall E_t$ that is not shown in Figure 3–15 is the well-formedness of term t, *viz.* that t is well-formed of the same type as variable x in the context of (free) variables that is effective at the point where the rule is applied.

As has been said one of the reasons for introducing a natural deduction presentation of lax logic is that the process of extracting constraint terms can be nicely explained: If we forget the formulae and only note down the rule names then a natural deduction tree is nothing but a term of nested rule applications. This term can be written down in linear notation and then directly transformed into the constraint term as shown in Figure 3–16. The way to apply this translation should be obvious.

Not having formalized the exact relationship between the sequent style presentation of lax logic of Section 3.1.3 and the natural deduction trees presented here we cannot, of course, state and prove the equivalence of translating a natural deduction tree as defined through Figure 3–16 and the extraction of constraint terms defined in Section 3.2. The natural deduction approach introduced in this section, therefore, should be regarded merely as a convenient shorthand for presenting the examples in the following Chapter 4. So, rather than proving formal correctness we contend ourselves with suggesting, via two examples, that reading a natural deduction tree as a λ-term in the way defined here results in exactly the same well-formed constraint term that one would get if the extraction of Section 3.2 had been applied starting from the corresponding sequent style derivation.

As the first example the derivation of rule $\Diamond L$ from $\Diamond F$ and $\Diamond M$ is chosen. The sequent style proof was given in Figure 3–13. In natural deduction form this proof reads as follows:

$$\dfrac{\dfrac{\Diamond M \qquad \dfrac{\Diamond N}{\Diamond\Diamond N}\Diamond M}{\Diamond N}\Diamond F_w}{\begin{array}{c}\vdots t\\ w : M \checkmark\end{array}}$$

The ellipsis $\cdots$ stands for an arbitrary derivation, named t, of $\Diamond N$ from M, *i.e.* for the sequent $\Gamma, M \vdash_\Delta \Diamond N$. This derivation, or natural deduction really is a parameter to the above tree. The variable w in the tree serves two purposes: In the natural deduction tree it uniquely identifies the assumption $w : M$ that is discharged by the application of rule $\Diamond F_w$, discharging being indicated by $\checkmark$. For the constraint term underlying the tree w is a variable of type $|M|$ of which $M^\sharp\, w$ may be assumed. It will appear free in the term represented by derivation t and it is bound (rather: eliminated) by the term forming operation underlying rule $\Diamond F_w$.

If the proof tree is written out in linear notation we get

$$\Diamond M\, \Diamond F_w(z, t).$$

The rule $\Diamond F_w$ has a left and a right sub-derivation, represented in the term by the pair (z,t). The left subtree is already a leaf, *i.e.* an assumption, so we put in a variable z in its place. Variable z is free in the term $\Diamond M\, \Diamond F_w(z,t)$ and because it is representing the assumption $\Diamond M$ it must be of type $\Omega^* \times |M|$. As explained the right subtree is represented by t with w as a free variable. Now we apply the translations of Figure 3–16 to this term and obtain

$$\begin{aligned}\Diamond M\, \Diamond F_w(z,t) &\equiv \Diamond M\,(\pi_1 z, t\{\pi_2 z/w\}) \\ &\equiv (\pi_1\pi_2(\pi_1 z, t\{\pi_2 z/w\}) @ \pi_1(\pi_1 z, t\{\pi_2 z/w\}),\\ &\qquad \pi_2\pi_2(\pi_1 z, t\{\pi_2 z/w\}))\end{aligned}$$

which is exactly the same term as the one extracted from the sequent style derivation in Figure 3–13. The names for variables z and w, as well as sub-derivation t were deliberately chosen so as to match Figure 3–13.

The second example takes up the derivation of Figure 3–14. In natural deduction style the derivation becomes

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash (\forall x^{\alpha}.\Diamond\iota\phi) \supset \forall y^{\beta}.\Diamond\iota(\phi\{f_l\}\vee\phi\{f_r\})}{\forall y^{\beta}.\Diamond\iota(\phi\{f_l\}\vee\phi\{f_r\})}\supset I_z}{\Diamond\iota(\phi\{f_l\}\vee\phi\{f_r\})}\forall I_y}{\dfrac{\Diamond\iota\phi\{f_i\}}{z:\forall x^{\alpha}.\Diamond\iota\phi\,\checkmark}\forall E_{f_i} \quad \dfrac{\iota(\phi\{f_l\}\vee\phi\{f_r\})}{v:\iota\phi\{f_i\}\,\checkmark}\iota}\Diamond F_v$$

Note, the proof of sequent $\phi\{f_i\} \vdash_y \phi\{f_l\} \vee \phi\{f_r\}$ contained in the sequent derivation of Figure 3–14 is not part of the natural deduction tree. As mentioned before, the structure of derivations in the base logic is not of interest as far as the constraint terms are concerned and thus will never be written out in the examples. Also, it can be seen that the typing $y \vdash f_i : \alpha$ is not carried over to the natural deduction tree. The above tree reads in linear notation:

$$\supset I_z \forall I_y \Diamond F_v(\forall E_{f_i}\, z\ ,\ \iota v).$$

Now we translate it step-by-step into a proper λ-term according to Figure 3–16. We remark that the translations are understood as syntactic identities, so the

order in which they are applied does not matter.

$$\begin{aligned}
&\supset I_z \forall I_y \Diamond F_v(\forall E_{f_i}\, z\ ,\ \iota\, v) \\
&\qquad \equiv\ \supset I_z \forall I_y \Diamond F_v(\forall E_{f_i}\, z\ ,\ *) \\
&\qquad \equiv\ \supset I_z \forall I_y \Diamond F_v(z\, f_i\ ,\ *) \\
&\qquad \equiv\ \supset I_z \forall I_y\, (\pi_1(z\, f_i)\, ,\ *) \\
&\qquad \equiv\ \lambda z. \forall I_y\, (\pi_1(z\, f_i)\, ,\ *) \\
&\qquad \equiv\ \lambda z. \lambda y. (\pi_1(z\, f_i)\, ,\ *).
\end{aligned}$$

Comparing this with the lower part of Figure 3–14 one finds again that one ends up with exactly the same term as the constraint extraction process for the sequent derivation. Of course the exact syntactic identity depends in this case on the choice of the bound variables z, y.

$$\frac{}{true}\,trueI \qquad \frac{M}{false}\,falseE \qquad \frac{M \quad N}{M \wedge N}\wedge I \qquad \frac{M \wedge N}{M}\wedge E_l \qquad \frac{M \wedge N}{N}\wedge E_r$$

$$\frac{M}{M \vee N}\vee I_l \qquad \frac{N}{M \vee N}\vee I_r \qquad \frac{M_1 \vee M_2 \quad \begin{array}{c}[x_1 : M_1\,\checkmark]\\ N\\ \vdots\end{array} \quad \begin{array}{c}[x_2 : M_2\,\checkmark]\\ N\\ \vdots\end{array}}{N}\vee E_{x_1,x_2}$$

$$\frac{M}{\forall x.\, M}\forall I_x \qquad \frac{\forall x.\, M}{M\{t/x\}}\forall E_t \qquad \frac{\exists x.\, M \quad \begin{array}{c}[y : M\,\checkmark]\\ N\\ \vdots\end{array}}{N}\exists E_y \qquad \frac{M\{t/x\}}{\exists x.\, M}\exists I_t$$

$$\frac{\iota\psi}{\iota\phi_1 \;\cdots\; \iota\phi_k}\,\iota,\ \textit{side condition: } \phi_1, \ldots, \phi_k \vdash_\Delta \psi$$

$$\frac{M\{t/x\} \quad M\{s/x\}}{\iota(s = t)}\,Subst \qquad \frac{M \supset N \quad \begin{array}{c}[x : M\,\checkmark]\\ N\\ \vdots\end{array}}{N}\supset I_x \qquad \frac{N \quad M \supset N}{M}\supset E$$

$$\frac{\Diamond M \quad \Diamond N}{\Diamond N}\Diamond L_x \qquad \frac{M}{\Diamond M}\Diamond I \qquad \frac{\Diamond\Diamond M}{\Diamond M}\Diamond M \qquad \frac{\Diamond M \quad \begin{array}{c}[x : M\,\checkmark]\\ N\\ \vdots\end{array}}{\Diamond N}\Diamond F_x$$

$$\frac{M\{0/n\} \quad M\{succ\,n/n\}}{\forall n.\, M}\,NatInd_{n,x} \qquad \frac{M\{[\,]/l\} \quad M\{h :: l/l\}}{\forall l.\, M}\,ListInd_{h,l,x}$$

Figure 3–15: Natural Deduction Rules of Lax Logic

$$
\begin{aligned}
\mathit{trueI} &\equiv * \\
\mathit{falseE}(a) &\equiv \Box\, a \\
\wedge I(a,b) &\equiv (a,b) \\
\wedge E_l(a) &\equiv \pi_1\, a \\
\wedge E_r(a) &\equiv \pi_2\, a \\
\vee I_l(a) &\equiv \iota_1\, a \\
\vee I_r(a) &\equiv \iota_2\, a \\
\vee E_{x_1,x_2}(a,b_1,b_2) &\equiv \mathit{case}_{x_1,x_2}(a,b_1,b_2) \\
\forall I_x(a) &\equiv \lambda x.\, a \\
\forall E_t(a) &\equiv a\, t \\
\exists E_y(a,b) &\equiv b\{\pi_1 a/x\}\{\pi_2 a/y\} \\
\exists I_t(a) &\equiv (t,a) \\
\iota(a_1,\ldots,a_k) &\equiv * \\
\supset I_x(a) &\equiv \lambda x.\, a \\
\supset E(a,b) &\equiv a\, b \\
\Diamond L_x(a,b) &\equiv (\pi_1(b\{\pi_2 a/x\})\ @\ \pi_1 a\ ,\ \pi_2(b\{\pi_2 a/x\})) \\
\Diamond I(a) &\equiv ([\,],a) \\
\Diamond M(a) &\equiv ((\pi_1\pi_2 a)@(\pi_1 a), \pi_2\pi_2 a) \\
\Diamond F_x(a,b) &\equiv (\pi_1 a, b\{\pi_2 a/x\}) \\
\mathit{Subst}(a,b) &\equiv b \\
\mathit{NatInd}_{n,x}(a,b) &\equiv \mathit{natrec}\,(a, \lambda n.\, \lambda x.\, b) \\
\mathit{ListInd}_{h,l,x}(a,b) &\equiv \mathit{listrec}\,(a, \lambda h.\, \lambda l.\, \lambda x.\, b)
\end{aligned}
$$

Figure 3–16: Translation of Natural Deduction Trees into Constraint Terms

Chapter 4

Application Examples

With lax logic in place we are now going to demonstrate its use on a number of verification examples exhibiting different standard abstractions with different characteristic constraints. We will put to work the techniques introduced abstractly in Section 3.2.1 for the particular notion of constraint $(\Omega^*, [\,], \bigcirc)$.

The examples of Section 4.1 are non-trivial in the sense that they involve induction proofs at least once in each case. Section 4.1 is placed at a higher level of behavioural abstraction focusing on data abstraction for simple functional programs. Section 4.2 expands in more detail on an example of synchronous circuit design with the main abstraction mechanism being timing abstraction.

4.1 Decrementor, Incrementor, and Factorial

The examples presented in this section are the decrementor, incrementor, and factorial function. Some attention is paid to methodological aspects of designing these functions using lax logic. The design is structured into the phases *modularization*, *realization*, and *composition* and in each of these phases *constraint extraction* and *constraint analysis* may be performed separately. The decrementor essentially is a decomposition exercise, the incrementor focuses on realization and finally all of the three phases will be relevant for the factorial example.

Incrementor, decrementor and factorial are designed at *the abstract level* over the domain of natural numbers. They are *modularized* into subcomponents via primitive recursion. For the incrementor, of course, this is trivial. For the decrementor this decomposition proof systematically introduces a lower bound as a constraint on inputs. In the case of the factorial it will be seen how composing proofs propagates input constraints of subcomponents to a constraint for the composite design.

Another step in the design of the factorial consists of *realizing* the incrementor (a subcomponent for the factorial) at the *concrete level* of finite bit-vectors which brings up an upper bound as the input constraint.

Further, incrementor and decrementor implementations will be *composed* to obtain the identity function as a simple example of how computations in the lambda calculus of constraints can serve to simplify constraints. We will observe that the lower bound "automatically" vanishes through constraint computation.

The associated proofs exhibit three different ways of manipulating constraints. They are induction proofs over natural numbers and the length of bit-vectors and differ in the way constraints enter and evolve in the course of induction. For the incrementor (at the concrete level) and decrementor (at the abstract level) constraints are introduced only in the base case of the induction. In contrast, the modularization proof for the factorial introduces a fresh constraint at each induction stage, but does not require a constraint to prove the base case. As to the way constraints evolve, both the incrementor and factorial pass on the constraint from one stage to the next after having modified it appropriately, while in the decrementor example no constraint has to be propagated at all.

A word on presentation: We will allow ourselves to be fairly loose with proofs and constructions that pertain to the base logic. One one side this is mandatory to cut down our discussion to a sensible size and to avoid being unreasonably formalistic. On the other side this is justified by the fact that proofs and propositions of the base logic are irrelevant as far as the construction and manipulation

of constraints is concerned. The base logic, as it were, is blanked by the process of constraint extraction. Thus, it is only that part of a formal argument that takes places in (the extension of) lax logic that we need to be rigorous about. Technically speaking, proofs in the base logic are treated as implicit side conditions on the embedding rule ι.

A word on notation: Any small caps Roman letter can serve as a variable, *e.g.* i, m, f, x, y, *etc.* are going to be used as object variables. In contrast composite names like *fac* and *cnt* are always meta-variables or abbreviations for (composite) terms. Finally, three different notions of equality are going to occur side-by-side that should be carefully distinguished: $\equiv$ is meta-logical, *i.e.* syntactical identity; $=$ denotes provable equality (an atomic proposition of the base logic that may represent computational normalization on the primitive type Ω); $\cong$ is the logical equivalence of propositions, *i.e.* bi-implication.

4.1.1 Designing the Decrementor

Let us begin by taking up our running example introduced in Section 2.2, which is a very simple decomposition exercise at the abstract level of natural numbers. The task is to implement and verify a decrementing function $dec : \mathbf{N} \Rightarrow \mathbf{N}$ for natural numbers obeying the lax specification

$$\forall n.\, \Diamond\, \iota\, (succ(dec\, n) = n). \tag{4.1}$$

In the following we will pick an approximate implementation for the decrementor and prove it correct *wrt.* specification (4.1). It is shown how an input constraint equivalent to $n \geq 1$ can be extracted from the proof by the method of Section 3.2. We point out that it is not *exactly* the predicate $n \geq 1$ that is extracted but a characteristic function $\chi_{n \geq 1} : \mathbf{N} \Rightarrow \Omega$ which is better than $\lambda n.\, n \geq 1$ in the sense that it exhibits more computational behaviour. This will be demonstrated in Section 4.1.2 below.

Preliminaries

The central data structure are the natural numbers

$$(\mathrm{N}, 0 : \mathrm{N}, succ : \mathrm{N} \Rightarrow \mathrm{N})$$

with recursion operator *natrec*, *i.e.* for each type τ, element a of type τ and function $f : \mathrm{N} \Rightarrow \tau \Rightarrow \tau$ the term $natrec\,(a, f) : \mathrm{N} \Rightarrow \tau$ can be formed, for which the equations

$$\begin{aligned} natrec(a, f)\,0 &= a \\ natrec(a, f)\,(succ\,n) &= f\,n\,(natrec(a, f)\,n) \end{aligned}$$

hold. Further, we will need the standard induction rule for natural numbers

$$\frac{\forall n^{\mathrm{N}}.\, M \qquad \begin{array}{c} M\{0/n\} \\ \vdots \\ x : M \checkmark \end{array} \qquad M\{succ\,n/n\}}{}\; NatInd_{n,x}$$

with the side condition that variable n may not be free in any assumption on which $M\{succ\,n/n\}$ depends other than $x : M$. Recall that x is a label which uniquely specifies the assumption M discharged by applying rule $NatInd_{n,x}$, while discharging is indicated by the symbol $\checkmark$. When translating this rule into a constraint term, $x : M$ is read as saying that variable x is of type $|M|$ such that $M^{\bullet}x$, or equivalently that x is a constraint term for M. It is important to note that the induction rule is a rule of lax logic for which constraint extraction is well-declared, so M may be an arbitrary well-formed formula containing $\Diamond$ and ι operators.

At the level of the base logic we will assume the standard natural number arithmetic, *i.e.* Peano's axioms as a global theory in which the discussions take place. The usual convention of writing the natural numbers

$$1 \stackrel{df}{\equiv} succ(0) \qquad 2 \stackrel{df}{\equiv} succ(succ(0)) \qquad 3 \stackrel{df}{\equiv} succ(succ(succ(0))) \qquad \cdots$$

applies.

In this and in the examples to follow we will make use of the fact that the object language of the base logic is a sufficiently rich lambda calculus, so that arithmetical operations like addition, multiplication, *mod*, *div*, *etc.* can be defined in the usual way. In order to keep the examples reasonably simple we will make free use of such operations and their algebraic properties without explicitly justifying the assumptions by deriving everything from scratch. In principle, however, everything can be nailed down rigorously in the base logic.

Verifying the Decrementor

The decrementor that we are going to verify maps zero back to itself, *i.e.* it solves the recursive definition

$$dec\,0 = 0 \qquad dec\,(succ\,n) = n$$

It is obtained via primitive recursion as the term

$$dec \stackrel{df}{\equiv} natrec\,(0, \lambda n.\,\lambda x.\,n)$$

with n, x variables of type N. A simple proof of specification (4.1) from the hypothesis *let's-not-bother* $\equiv \forall z.\,\Diamond_\iota z$, *i.e.* a derivation of the sequent

$$let's\text{-}not\text{-}bother \vdash \forall n.\,\Diamond_\iota(succ(dec\,n) = n) \tag{4.2}$$

is given in Figure 4–1. The purpose of hypothesis *let's-not-bother* will become clear below.

$$\dfrac{\dfrac{\dfrac{}{\forall n.\,\Diamond_\iota(succ(dec\,n) = n)}}{\dfrac{\Diamond_\iota(succ(dec\,0) = 0)}{\forall z.\,\Diamond_\iota z}\forall E_{succ(dec\,0)=0}} \quad \dfrac{\dfrac{\dfrac{}{\Diamond_\iota(succ(dec(succ\,n)) = succ\,n)}}{\iota(succ(dec(succ\,n)) = succ\,n)}\Diamond I}{\iota}\; NatInd_{n,y}$$

Figure 4–1: Derivation which Verifies the Decrementor Function

We are interested in that part of the proof where constraint manipulation takes place. Consequently, Figure 4–1 only shows the derivation in lax logic. The

proof exercises natural induction (rule $NatInd_{n,y}$) with the base case (left subtree) being derived directly from the hypothesis *let's-not-bother*. This is the only choice possible as the base case amounts to the correctness of the decrementor for input 0 which cannot be obtained in the usual strict sense. In the right subtree, which amounts to the induction step, rule ι depends (side condition) on a subproof in the base logic of the sequent

$$\vdash_n succ(dec(succ\, n)) = succ\, n$$

Such a proof can easily be found by induction over n and some equational reasoning using the definition of *dec*. The actual structure of this proof in the base logic is irrelevant and not given here, although of course a great deal of the verification is done there. Notice that the induction step does not use the induction hypothesis $\Diamond\iota(succ(dec\, n) = n)$ and also does not refer to *let's-not-bother*. The variable y which appears in rule $NatInd_{n,y}$ of Figure 4–1 refers to the induction hypothesis which itself is not written down as an assumption.

Constraint Analysis for the Decrementor

Let us analyze the constraint information contained in the derivation of Figure 4–1 of sequent (4.2) along the lines set out in Section 3.2.1 of Chapter 3. We compute constraint types as

$$\begin{aligned} |\forall z.\Diamond\iota z| &\equiv \Omega \Rightarrow (\Omega^* \times \mathbf{1}) \\ |\forall n.\Diamond\iota(succ(dec\, n) = n)| &\equiv \mathbf{N} \Rightarrow (\Omega^* \times \mathbf{1}). \end{aligned}$$

Let w in the following be a variable of type $\Omega \Rightarrow (\Omega^* \times \mathbf{1})$. We expect to extract a well-formed constraint term D of type $\mathbf{N} \Rightarrow (\Omega^* \times \mathbf{1})$ with w as its free variable such that

$$(\forall z.\Diamond\iota z)^\# w \vdash_w (\forall n.\Diamond\iota(succ(dec\, n) = n))^\# D \tag{4.3}$$

is derivable in the base logic. Working out the constraint predicates

$$\begin{aligned} (\forall z.\Diamond\iota z)^\# w &\equiv \forall z.\, z^{\pi_1(wz)} \\ (\forall n.\Diamond\iota(succ(dec\, n) = n))^\# D &\equiv \forall n.\, (succ(dec\, n) = n)^{\pi_1(Dn)} \end{aligned}$$

we find that (4.3) means

$$\forall z.\, z^{\pi_1(wz)} \vdash_w \forall n.\, (succ(dec\, n) = n)^{\pi_1(Dn)}. \tag{4.4}$$

Applying the idea of Section 3.2.1 we specialize w to the (closed) term $? \equiv \lambda z.\, ([z], *)$ so that the hypothesis of (4.4) becomes provably equal to $\forall z.\, z \supset z$, which is trivially provable, and hence (4.4) can be simplified to

$$\vdash \forall n.\, (succ(dec\, n) = n)^C$$

or, to bring into play function $\sqcap : \Omega^* \Rightarrow \Omega$ (*cf.* Sec. 3.1.2), to the sequent

$$\vdash \forall n.\, (\sqcap C) \supset succ(dec\, n) = n$$

where C of type Ω^* abbreviates the term $\pi_1(D\{?/w\}n)$. Thus, proposition list C (or proposition $\sqcap C$), which has n as free variable, is the constraint on the input n under which *dec* satisfies its specification $succ(dec\, n) = n$. Note that C is specific to the particular proof of Figure 4–1. A different proof might result in a different proposition list C. Our plan now is to work out what C looks like and to verify that it is equivalent to $n \geq 1$ which is the constraint we intuitively expect.

In order to extract the constraint term D from the derivation of Figure 4–1 we work from the natural deduction proof as explained in Section 3.3 and consider the derivation as a tree and, in linear notation, a term of rule applications:

$$\dfrac{\dfrac{NatInd_{n,y}}{\forall E_{succ(dec\,0)=0}}}{w} \qquad \dfrac{\Diamond I}{\iota} \qquad\qquad NatInd_{n,y}(\forall E_{succ(dec\,0)=0}\, w\ ,\ \Diamond I\ \iota)$$

where variable w refers to the hypothesis *let's-not-bother*. Constraint term D is obtained from this as follows (*cf.* Table 3–16 in Sec. 3.3):

$$\begin{aligned} D &\equiv NatInd_{n,y}(\forall E_{succ(dec\,0)=0}\, w\ ,\ \Diamond I\ \iota) \\ &\equiv natrec\,(\forall E_{succ(dec\,0)=0}\, w\ ,\ \lambda n.\, \lambda y.\, \Diamond I\ \iota) \\ &\equiv natrec\,(w(succ(dec\,0) = 0)\ ,\ \lambda n.\, \lambda y.\, ([\,], \iota)) \\ &\equiv natrec\,(w(succ(dec\,0) = 0)\ ,\ \lambda n.\, \lambda y.\, ([\,], *)) \end{aligned}$$

Now we can evaluate the proposition list $C \equiv \pi_1(D\{?/w\}n)$. For $n \equiv 0$ we get

$$\begin{aligned} C\{0/n\} &\equiv \pi_1(D\{?/w\}0) \\ &\equiv \pi_1(natrec\,(?(succ(dec\,0) = 0)\ ,\ \lambda n.\,\lambda y.\,([\,]\ ,\ *))\,0) \\ &= \pi_1(natrec\,(([succ(dec\,0) = 0], *)\ ,\ \lambda n.\,\lambda y.\,([\,]\ ,\ *))\,0) \\ &= \pi_1([succ(dec\,0) = 0], *) \\ &= [succ(dec\,0) = 0] \end{aligned}$$

where $=$ means provable equality in the base logic. Thus, the constraint C for the base case $n \equiv 0$ consists of the base case itself. This is not surprising since the base case was discharged by resorting to *let's-not-bother*, and so no work was done at all. Removing the hypothesis *let's-not-bother* in the way described via constraint extraction, then, makes the base case pop up again as the constraint. For $n \equiv succ\,k$ one computes

$$\begin{aligned} C\{succ\,k/n\} &\equiv \pi_1(D\{?/w\}(succ\,k)) \\ &\equiv \pi_1(natrec\,(?(succ(dec\,0) = 0)\ ,\ \lambda n.\,\lambda y.\,([\,]\ ,\ *))\,(succ\,k)) \\ &= \pi_1(natrec\,(([succ(dec\,0) = 0], *)\ ,\ \lambda n.\,\lambda y.\,([\,]\ ,\ *))\,(succ\,k)) \\ &= \pi_1((\lambda n.\,\lambda y.\,([\,]\ ,\ *))\,k\,(natrec\,(([succ(dec\,0) = 0], *)\ , \\ &\qquad\qquad \lambda n.\,\lambda y.\,([\,]\ ,\ *))\,k)) \\ &= \pi_1([\,]\ ,\ *) \\ &= [\,] \end{aligned}$$

Thus, the constraint list C for all the other (induction) cases is empty which reflects the fact that the induction step does not refer to *let's-not-bother* for solving subgoals.

The following proposition validates both the extraction process and the proof as it shows that the extracted constraint predicate $\sqcap C$ is essentially what we would hope for, *viz.* the weakest input constraint for the chosen implementation *dec*:

Proposition 4.1.1 $\sqcap C \cong n \geq 1$.

Proof: The proof proceeds by induction on n. For $n \equiv 0$ one has $\sqcap(C\{0/n\}) = \sqcap[succ(dec\,0) = 0] = \sqcap[1 = 0] = true \wedge 1 = 0$ which is equivalent to $0 \geq 1$ as both conditions are false (Peano's axiom $succ\,n \neq 0$ tacitly assumed). For $n \equiv succ\,k$ one computes $\sqcap(C\{succ\,k/n\}) = \sqcap[\,] = true$ which is equivalent to $succ\,k \geq 1$ as both propositions are true (again Peano's axioms assumed). ∎

If one wishes to do so, Proposition 4.1.1 now justifies replacing the extracted constraint term $D : \mathbf{N} \Rightarrow \Omega^* \times \mathbf{1}$ by the simpler term

$$D' \stackrel{df}{=} \lambda n.\,([n \geq 1], *)$$

and

$$(\forall z.\,\Diamond\iota z)^* w \vdash_w (\forall n.\,\Diamond\iota(succ(dec\,n) = n))^* D'$$

is still derivable. However, D has the advantage that by simple β-normalization the constraint $[\,] : \Omega^*$ results, when it is used for $n = succ\,k$, which is more useful than the constraint $[succ\,k \geq 1]$ obtained from D'. In other words, the extracted lambda-term D has potential computational behaviour while the processed D' does not. An example of how this can be used will be discussed in Section 4.1.2.

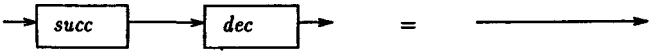
* * *

To sum up the constraint encapsulated in the proof of the decrementor defines a non-trivial condition only for $n \equiv 0$. What is going on, basically, is that the induction skips the base case $n \equiv 0$ trading it for a constraint, while the induction step does the main job of verifying from $n \equiv 1$ onwards. It has been shown that the constraint extracted by the methods defined in Section 3.2 is equivalent to the condition $n \geq 1$.

Notice, the evaluation of the constraint predicate $\sqcap C$ carried through above, for particular closed terms n, is algorithmical, *viz.* $\equiv$ reasoning basically is unfolding definitions and $=$ reasoning is applying β-reductions only. The non-algorithmical part is the creative step of coming up with the predicate $n \geq 1$, and the induction argument in the proof of Proposition 4.1.1.

4.1.2 Composing Incrementor and Decrementor

In this section we briefly illustrate the point made above that the constraint predicate $\sqcap C$ extracted for the decrementor, or $D\{?/w\}$ for that matter, has computational behaviour, and that this makes it more useful than the 'canonical' predicate $n \geq 1$ which is nice to look at but also rather rigid. We show that if we precompose the decrementor with an ideal incrementor, *i.e.* the successor function, to obtain the identity



then resulting input constraint computes away simply by applying β-equalities on the constraint term. This substantiates the claim that an operational interpretation of constraint terms can be used to automatically manipulate and even simplify constraints.

The statement that feeding the output of the successor function into the decrementor yields — up to possible constraints — the identity, is expressed by the sequent

$$\forall n.\Diamond\iota(succ(dec\,n) = n) \;\vdash\; \forall n.\Diamond\iota(dec(succ\,n) = n).$$

The simple derivation of this sequent is presented in Figure 4–2, where the application of rule ι is justified by assuming the Peano axioms in $\mathcal{B}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall n.\Diamond\iota(dec(succ\,n) = n)}{\Diamond\iota(dec(succ\,n) = n)}\forall I_n}{\Diamond\iota(succ(dec(succ\,n)) = succ\,n)}\Diamond F_y}{\forall n.\Diamond\iota(succ(dec\,n) = n)}\forall E_{succ\,n} \quad \dfrac{\iota(dec(succ\,n) = n)}{y : \iota(succ(dec(succ\,n)) = succ\,n)\;\surd}\iota$$

Figure 4–2: Derivation for Composition of Incrementor and Decrementor

From this derivation we extract a well-formed constraint term ct of type $|\forall n.\Diamond\iota(dec(succ\,n) = n)| \equiv \mathbf{N} \Rightarrow \Omega^* \times \mathbf{1}$ with a free variable z of type

$|\forall n.\, \Diamond\, \iota\,(succ(dec\, n) = n)| \equiv \mathbf{N} \Rightarrow \Omega^* \times \mathbf{1}$ such that the sequent

$$(\forall n.\, \Diamond\, \iota\,(succ(dec\, n) = n))^{\#}\, z \;\; \vdash_z \;\; (\forall n.\, \Diamond\, \iota\,(dec(succ\, n) = n))^{\#}\, ct$$

is derivable in $\mathcal{B}$, which after expanding the definition of $(\cdot)^{\#}$ gives the equivalent

$$\forall n.\, (succ(dec\, n) = n)^{\pi_1\,(z\,n)} \;\; \vdash_z \;\; \forall n.\, (dec(succ\, n) = n)^{\pi_1\,(ct\,n)}. \qquad (4.5)$$

Thus, the expression ct translates a constraint term z for the decrementor into a constraint term $\pi_1(ct\, n)$ for the resulting identity. We want to show that if we instantiate z by the particular constraint term $D\{?/w\}$ computed for the decrementor in the previous section, then the resulting constraint becomes trivial, *i.e.*

$$\pi_1\,(ct\{D\{?/w\}/z\}\, n) \;\; = \;\; []$$

and further that this equation is provable just by applying β-equalities.

The term ct is computed from the derivation tree as follows (*cf.* Fig. 3–16 in Sec. 3.3):

$$\begin{aligned} ct \;\; &\equiv \;\; \forall I_n \Diamond F_y\,(\forall E_{succ\, n}\, z\,,\; \iota y) \\ &\equiv \;\; \forall I_n \Diamond F_y\,(z\,(succ\, n)\,,\; *) \\ &\equiv \;\; \forall I_n\,(\pi_1(z\,(succ\, n))\,,\; *) \\ &\equiv \;\; \lambda n.\,(\pi_1(z\,(succ\, n))\,,\; *) \end{aligned}$$

Hence, substituting $D\{?/w\}$ for z (D is defined in the previous section) gives

$$\begin{aligned} &\pi_1\,(ct\{D\{?/w\}/z\}\, n) \\ &\quad \equiv \;\; \pi_1\,(\lambda n.\,(\pi_1((D\{?/w\})\,(succ\, n))\,,\; *)\, n) \\ &\quad \equiv \;\; \pi_1\,((\lambda n.\,(\pi_1((natrec\,(?\,(succ(dec\, 0) = 0)\,,\; \lambda n.\, \lambda y.\,([]\,,\; *)))\,(succ\, n))\,,\; *))\, n) \\ &\quad = \;\; \pi_1\,((\lambda n.\,(\pi_1((\lambda n.\, \lambda y.\,([]\,,\; *))\, n\,(natrec\,(\cdots, \cdots)))\,,\; *))\, n) \\ &\quad = \;\; \pi_1\,((\lambda n.\,(\pi_1([]\,,\; *)*))\, n) \\ &\quad = \;\; \pi_1\,((\lambda n.\,([]\,,\; *))\, n) \\ &\quad = \;\; \pi_1\,([]\,,\; *) \\ &\quad = \;\; [] \end{aligned}$$

The fourth line is by β-equality of *natrec*. The fifth and seventh line by β-equality for λ, and finally sixth and eights line by β-equality for pairing. Notice that all applications of *beta*-equality actually are *β-reductions*, so that the above chain of equations would completely be automated by an operational implementation of the object language of $\mathcal{B}$ as an ordinary lambda-calculus.

That the computed term [] in fact is the constraint for the resulting identity function, *viz.* a proof in $\mathcal{B}$ of $\forall n.\,(dec(succ\,n) = n)^{[]}$, can be derived from (4.5) and the result that $C \equiv \pi_1(D\{?/w\}\,n)$ is a constraint for the decrementor, *viz.* that there is a proof of $\forall n.\,(succ(dec\,n) = n)^{C}$

* * *

4.1.3 Designing the Incrementor

Let us consider another simple verification example which, as far as the business of constraints is concerned, is different in character from the previous one and slightly more complex. We set ourselves the task of realizing the abstract successor function at the concrete level of finite bit-vectors. The resulting combinatory circuit is an *incrementor*. In this example the constraint crops in as an *overflow* constraint compensating for the imprecision associated with representing natural numbers by bit-vectors.

Preliminaries

We start by introducing the general setting and the abstraction and realization mappings connecting up the two levels of data abstraction. Some simplifying assumptions need to be explained, in particular concerning the formalization of finite bit-vectors and related operations on bit-vectors.

The type of individual bits will be denoted by $\mathbf{B}$, and, for simplicity, is treated as the subset $\{0,1\} \subset \mathbf{N}$ of natural numbers. In practice this means that at the informal level we may use natural number arithmetic for manipulating bits,

provided we stay within the specified range. It would be more accurate, of course, to model bits by the type $\mathbf{1}+\mathbf{1}$ and to define the usual bit operations such as exclusive-or ($\oplus$) or conjunction ($\wedge$) via constructors and destructor of the sum type. Anyway, these operations, as we will assume here, are defined equally well on the subset $\{0,1\}$ in the obvious way. One can put $a \oplus b = (a+b) \, mod \, 2$ and $a \wedge b = (a \cdot b) \, mod \, 2$, for instance.

For modelling bit-vectors, then, we assume for each natural number $w : \mathbf{N}$, a type $\mathbf{B}^w$, elements of which behave like bit lists of length w. Specifically, there is a unique element $[\,]$, the empty bit-vector, of type $\mathbf{B}^0$, and an operator $::$ taking a single bit $b : \mathbf{B}$ and a bit vector $v : \mathbf{B}^w$ into the vector $b :: v$ of length $w+1$. Just as for ordinary lists we will write $[b_{w-1}, \ldots, b_1, b_0]$ as an abbreviation for $b_0 :: b_1 \cdots b_{w-1} :: [\,]$.

The formalization of bits and bit-vectors in the base logic deserves some explanation: The base logic as it stands does not have subset types or dependent types, so strictly speaking, there is no subset type $\mathbf{B} = \{0,1\} \subset \mathbf{N}$ or $\mathbf{B}^w = \{v : \mathbf{B}^* \mid \mathit{length\ of}\ v = w\} \subset \mathbf{B}^*$, and there is no type $\mathbf{B}^{\mathit{expr}}$ depending on the value of an expression *expr*. Consequently, there are no well-formed formulae like

$$\forall c^{\mathbf{B}}.\, M \quad , \quad \forall w^{\mathbf{N}}.\forall z^{\mathbf{B}^w}.\, M.$$

Such formulae, however, can be used in the naive way if understood as abbreviations for

$$\forall c^{\mathbf{N}}.\, \iota(\mathit{is\text{-}bit}\,(c)) \supset M \quad , \quad \forall w^{\mathbf{N}}.\forall z^{\mathbf{N}^*}.\, \iota(\mathit{is\text{-}bit\text{-}vec}\,(w,z)) \supset M.$$

where $\mathit{is\text{-}bit}\,(c)$ and $\mathit{is\text{-}bit\text{-}vec}\,(w,z)$ are propositions characterizing the subsets $\mathbf{B} = \{0,1\} \subseteq \mathbf{N}$ and $\mathbf{B}^w \subseteq \mathbf{N}^w \subseteq \mathbf{N}^*$. In other words, $\mathbf{B}$ and $\mathbf{B}^w$ are mimicked by the nearest available super-types and the information thereby lost is shifted into the formulae. Also, for instance, a typing like

$$w^{\mathbf{N}}, z^{\mathbf{B}^{w+1}} \vdash f : \mathbf{B}^w$$

must be read as an abbreviation for the composite judgement

$$\begin{array}{rl} w^{\mathbf{N}}, z^{\mathbf{N}^*} & \vdash \quad f : \mathbf{N}^* \\ \textit{is-bit-vec}\,(w+1, z) & \vdash_{w,z} \quad \textit{is-bit-vec}\,(w, f). \end{array}$$

If this reading is applied systematically, then the elimination rule for universal quantification in a situation like

$$\frac{\forall z^{\mathbf{B}^w}.\, M \vdash_{w,x} M\{f/z\}}{w^{\mathbf{N}}, x^{\mathbf{B}^{w+1}} \vdash f : \mathbf{B}^w \quad w^{\mathbf{N}}, x^{\mathbf{B}^{w+1}}, z^{\mathbf{B}^w} \vdash M \text{ wff}}\forall E$$

really is an abbreviation for the derived rule

$$\frac{\forall z^{\mathbf{N}^*}.\, \iota(\textit{is-bit-vec}\,(w, z)) \supset M \vdash_{w,x} M\{f/z\}}{\begin{array}{l} \textit{is-bit-vec}\,(w+1, x) \vdash_{w,x} \textit{is-bit-vec}\,(w, f) \quad w^{\mathbf{N}}, x^{\mathbf{N}^*}, z^{\mathbf{N}^*} \vdash M \text{ wff} \\ w^{\mathbf{N}}, x^{\mathbf{N}^*} \vdash f : \mathbf{N}^* \end{array}}$$

We skip here the question of if and under what conditions this simulating of $\mathbf{B}$ and $\mathbf{B}^w$ is semantically equivalent.

For *translating* between both levels of the design, abstraction and realization mappings

$$\alpha_w : \mathbf{B}^w \Rightarrow \mathbf{N} \qquad \rho_w : \mathbf{N} \Rightarrow \mathbf{B}^w$$

are employed. We follow the convention of representing natural numbers in the dyadic number system and put

$$\alpha_w[b_{w-1}, \ldots, b_0] \stackrel{df}{=} \sum_{i=0}^{w-1} 2^i \cdot b_i \tag{4.6}$$

$$\rho_w\, n \stackrel{df}{=} [(n\, \textit{div}\, 2^{w-1})\, \textit{mod}\, 2\, , \ldots, \, (n\, \textit{div}\, 2^0)\, \textit{mod}\, 2]. \tag{4.7}$$

Note that ρ_w is a total function truncating naturals in case they are too big to fit within the space of w bits. We will not be more formal so as to define both mappings explicitly by definite terms of the object language of lax logic, *viz.* via the recursion operators for lists and natural numbers. For the sake of simplicity

we will assume that we are provided with terms for α_w, ρ_w for which the equations

$$\begin{aligned} \alpha_0[\,] &= 0 \\ \alpha_{w+1}[b_w, \ldots, b_1, b_0] &= 2 \cdot \alpha_w[b_w, \ldots, b_1] + b_0 \\ \rho_0\, n &= [\,] \\ \rho_{w+1}\, n &= n \bmod 2 :: \rho_w(n \operatorname{div} 2) \end{aligned}$$

are provable in the base logic. Although this may not be an entirely trivial requirement it does not pose any fundamental problem.

Verifying the Incrementor

The incrementor, as a bit-slice structure, is defined inductively along bit-length w. To indicate which bit-length we are dealing with w is used as a parameter, *i.e.* the w-bit incrementor is going to be a term

$$Inc_w : \mathbf{B}^w \Rightarrow \mathbf{B}^w.$$

We remind the reader of our explanations of how this typing is to be read in the absence of dependent types and subset types in the base logic. Inc_w will be built from w half-adders $Add : \mathbf{B} \times \mathbf{B} \Rightarrow \mathbf{B} \times \mathbf{B}$ where

$$\begin{aligned} Add(a, b) &\stackrel{df}{=} (Sum(a, b), Carry(a, b)) \\ Sum(a, b) &\stackrel{df}{=} a \oplus b \\ Carry(a, b) &\stackrel{df}{=} a \wedge b. \end{aligned}$$

A cascade of such one-bit half-adders as shown in Figure 4–3 yields a function $Add_w : \mathbf{B}^w \times \mathbf{B} \Rightarrow \mathbf{B}^w$ characterized by the equations

$$\begin{aligned} Add_{w+1}([b_w, \ldots, b_0], c_0) &= [b'_w, \ldots, b'_0] \\ b'_i &= Sum(b_i, c_i) \\ c_{i+1} &= Carry(b_i, c_i) \end{aligned}$$

or, more compactly by

$$Add_{w+1}(b_0 :: b,\, c_0) = Sum(b_0, c_0) :: Add_w(b, Carry(b_0, c_0)).$$

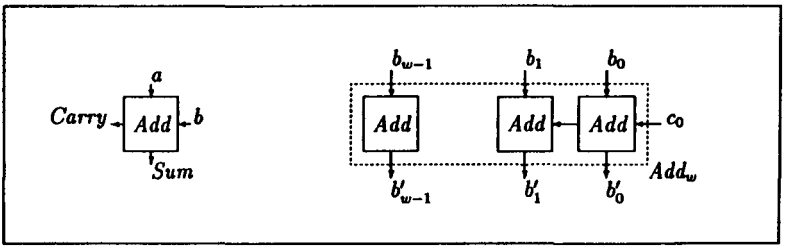


Figure 4–3: w-Bit Realization of Successor

It will be useful also to define a trivial 0-bit Adder $Add_0 : \mathbf{B}^0 \times \mathbf{B} \Rightarrow \mathbf{B}^0$, *viz.*

$$Add_0([\,], c_0) \;=\; [\,].$$

The incrementor is now obtained by specializing the carry input of Add_w to value 1, *i.e.*

$$Inc_w\, b \;\stackrel{df}{=}\; Add_w(b, 1)$$

for $w \geq 0$. Again, in order not to complicate things unnecessarily we do not want to be more formal and define the incrementor as some definite term in the object language of lax logic, *e.g.* via some explicit cascading combinator. We simply assume that Add_w is some specific term for which the equations above are provable in the base logic.

The verification goal expressing that Inc_w correctly realizes the successor operation is

$$\forall w^{\mathbf{N}}.\, \forall n^{\mathbf{N}}.\, \Diamond\iota((\alpha_w \circ Inc_w \circ \rho_w)\, n = succ\, n), \tag{4.8}$$

where α_w, ρ_w being the abstraction and realization mappings, and $\circ$ is the (associative) composition combinator $f \circ g = \lambda x.\, f(g\, x)$. Exploiting the fact that these mappings are defined by recursion on bit-length w the proof will proceed by natural induction on w.

Derivation 1 *The abstraction of the w-bit incrementor Inc_w satisfies the specification of the abstract incrementor under the hypothesis let's-not-bother* $\equiv \forall z.\, \Diamond\iota z$

for all w in $\mathbf{N}$*, i.e. there is a derivation of the sequent*

$$let's\text{-}not\text{-}bother \vdash \forall w^{\mathbf{N}}.\forall n^{\mathbf{N}}.\Diamond\iota((\alpha_w \circ Inc_w \circ \rho_w)\, n = succ\, n)$$

The global structure of the derivation, at the heart of which lies induction over w, is depicted in Figure 4–4.

$$\forall w.\forall n.\ \Diamond\iota((\alpha_w \circ Inc_w \circ \rho_w)\, n = succ\, n)$$

$$(P_1)$$

$$\frac{\forall w.\forall n.\forall c.\Diamond\iota(\alpha_w(Add_w(\rho_w\, n, c)) = n + c)}{\vdots \qquad \forall n.\forall c.\Diamond\iota(\alpha_{w+1}(Add_{w+1}(\rho_{w+1}\, n, c)) = n + c)}\ NatInd_{w,x}$$

see below $\qquad (P_2)$

$$x : \forall n.\forall c.\Diamond\iota(\alpha_w(Add_w(\rho_w\, n, c)) = n + c)\ \checkmark$$

$$\frac{\dfrac{\dfrac{\forall n.\forall c.\Diamond\iota(\alpha_0(Add_0(\rho_0\, n, c)) = n + c)}{\forall c.\Diamond\iota(\alpha_0(Add_0(\rho_0\, n, c)) = n + c)}\forall I_n}{\Diamond\iota(\alpha_0(Add_0(\rho_0\, n, c)) = n + c)}\forall I_c}{\forall z.\Diamond\iota z}\forall E_\gamma$$

$$\gamma \stackrel{df}{\equiv} \alpha_0(Add_0(\rho_0\, n, c)) = n + c$$

Figure 4–4: Global Structure of Derivation 1

The sub-derivation called (P_1) is reducing the goal to the slightly more general property

$$\forall w.\forall n.\forall c.\ \Diamond\iota(\alpha_w(Add_w(\rho_w\, n, c)) = n + c).$$

It says that Add_w is a 'stoppable' incrementor, *viz.* that it is incrementing if control input c has value 1, and otherwise ($c = 0$) is passing on the input value to the output. Sub-derivation (P_1), written out in Figure 4–5, essentially applies

the substitution rule *Subst* twice, the equations exploited being

$$succ\, n = n + 1$$

$$(\alpha_w \circ Inc_w \circ \rho_w)\, n = \alpha_w(Add_w(\rho_w\, n, 1)).$$

The proofs of both equations in the base logic are not difficult to establish. The rest of sub-derivation (P_1) is fairly simple and does not need to be explained.

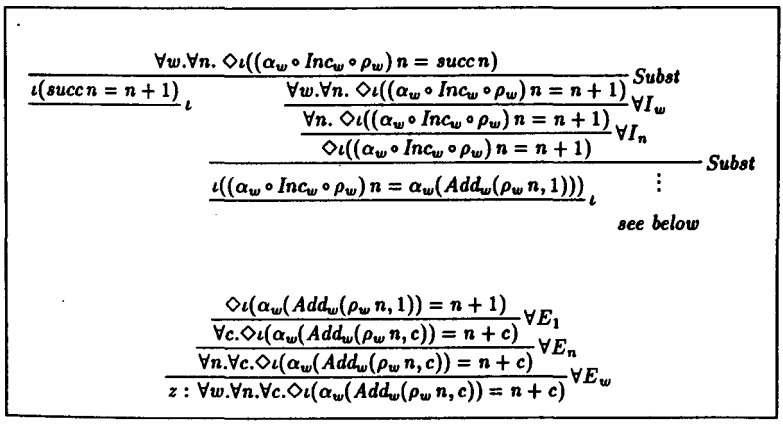


Figure 4–5: Sub-Derivation (P_1)

The more interesting part of Figure 4–4 is the induction consisting of rule $NatInd_{w,x}$. The left subtree of rule $NatInd_{w,x}$ is the base case of the induction, the right subtree is the induction step, which is abbreviated in Figure 4–4 by (P_2).

Just like in the case of the decrementor the base case is dealt with by using the global hypothesis *let's-not-bother*. It is here where a constraint is introduced into the proof. The subgoal that is solved using *let's-not-bother* instantiated with the equation $\gamma \stackrel{df}{\equiv} \alpha_0(Add_0(\rho_0\, n, c)) = n + c$, which by definition of α, *Add* and ρ is equivalent to $0 = n + c$. Although the condition $0 = n + c$ is satisfiable, if it is universally quantified over n and c, it is inconsistent with Peano arithmetic. In this sense the situation is the same as with the decrementor: The base case

without $\Diamond$ and ι is inconsistent (so it is not provable) but in lax logic it can be proven from the consistent hypothesis *let's-not-bother*.

The induction step, *viz.* sub-derivation (P_2), is shown in Figure 4–6. The fact that induction works, of course, owes to the regular bit-slice structure of the incrementor; it reduces the verification problem of the w-bit incrementor to the verification of a single 2-bit half-adder. Not surprisingly, as will be seen below, the correctness constraint for the w-bit incrementor constructed with the lax proof of Figure 4–4 essentially originates with a constraint for the 2-bit half-adder. The invariant, or property, of the half-adder driving the proof (side condition, or 'side proof' implicit in rule ι of Figure 4–6) is the law[1]

$$a + b = (a \oplus b) + 2 \cdot (a \wedge b). \tag{4.9}$$

From this it is easy to see that $a + b = a \oplus b$ *iff* $a \wedge b = 0$. Hence, the condition $a \wedge b = 0$ characterizes precisely how far the half-adder fails to implement addition, and consequently embodies the germ for the overflow constraint generated by the induction proof. We will be a little bit more precise about this below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall n.\forall c.\Diamond\iota(\alpha_{w+1}(Add_{w+1}(\rho_{w+1}\,n,c)) = n + c)}{\forall c.\Diamond\iota(\alpha_{w+1}(Add_{w+1}(\rho_{w+1}\,n,c)) = n + c)}\forall I_n}{\Diamond\iota(\alpha_{w+1}(Add_{w+1}(\rho_{w+1}\,n,c)) = n + c)}\forall I_c}{\begin{array}{c}\vdots\\ \textit{see below}\end{array}}\Diamond F_y \qquad \dfrac{\iota(\alpha_{w+1}(Add_{w+1}(\rho_{w+1}\,n,c)) = n + c)}{y:\ \iota(\alpha_w(Add_w(\rho_w(n\ div\ 2), n\ mod\ 2 \wedge c)) = n\ div\ 2 + (n\ mod\ 2 \wedge c))\ \checkmark}\iota$$

$$\dfrac{\dfrac{\Diamond\iota(\alpha_w(Add_w(\rho_w(n\ div\ 2), n\ mod\ 2 \wedge c)) = n\ div\ 2 + (n\ mod\ 2 \wedge c))}{\forall c.\Diamond\iota(\alpha_w(Add_w(\rho_w(n\ div\ 2), c)) = n\ div\ 2 + c)}\forall E_{n\ mod\ 2 \wedge c}}{\forall n.\forall c.\Diamond\iota(\alpha_w(Add_w(\rho_w\,n,c)) = n + c)}\forall E_{n\ div\ 2}$$

Figure 4–6: Step Case of Induction, Sub-Derivation (P_2)

[1] Another fundamental law exploited is $2 \cdot (n\ div\ 2) + (n\ mod\ 2) = n$.

Constraint Analysis for the Incrementor

Let us now, in all detail, work out the overflow constraint for the incrementor from Derivation 1,

$$\forall z.\, \Diamond \iota z \vdash \forall w.\, \forall n.\, \Diamond \iota((\alpha_w \circ Inc_w \circ \rho_w)\, n = succ\, n)$$

presented in Figures 4–4 – 4–6. The constraint types of both sides are

$$\begin{aligned} |\forall z.\, \Diamond \iota z| &\equiv \Omega \Rightarrow (\Omega^* \times 1) \\ |\forall w.\, \forall n.\, \Diamond \iota((\alpha_w \circ Inc_w \circ \rho_w)\, n = succ\, n)| &\equiv \mathbf{N} \Rightarrow \mathbf{N} \Rightarrow (\Omega^* \times 1). \end{aligned}$$

Thus, given a variable v of type $\Omega \Rightarrow (\Omega^* \times 1)$ we expect to extract a constraint term, say P, of type $\mathbf{N} \Rightarrow \mathbf{N} \Rightarrow (\Omega^* \times 1)$ with free variable v such that

$$(\forall z.\, \Diamond \iota z)^{\#}\, v \;\vdash_v\; (\forall w.\, \forall n.\, \Diamond \iota((\alpha_w \circ Inc_w \circ \rho_w)\, n = succ\, n))^{\#}\, P$$

which by definition of constraint predicates is the sequent

$$\forall z.\, z^{\pi_1(vz)} \;\vdash_v\; \forall w.\, \forall n.\, ((\alpha_w \circ Inc_w \circ \rho_w)\, n = succ\, n)^{\pi_1(Pwn)}.$$

Again, we specialize v to the closed term $? \equiv \lambda z.\, ([z], *)$ with the effect of trivializing the hypothesis and get

$$\vdash \forall w.\, \forall n.\, ((\alpha_w \circ Inc_w \circ \rho_w)\, n = succ\, n)^{\pi_1(P\{?/v\}wn)}. \tag{4.10}$$

Proposition list $C(w,n) \stackrel{df}{\equiv} \pi_1(P\{?/v\}wn)$ of type Ω^*, then, is the overflow constraint we are after. The whole proof consisting of Figures 4–4 – 4–6 reduced

to its underlying tree of rule applications is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P : Subst}{\iota \quad \forall I_w}}{\forall I_n}}{Subst}}{\iota \quad \forall E_1}}{\forall E_n}}{\forall E_w}}{P_3 : NatInd_{w,z}}}{\forall I_n \quad \dfrac{\dfrac{P_2 : \forall I_n}{\forall I_c}}{\diamond F_y}}}{\forall I_c}}{\forall E_\gamma \quad \dfrac{\forall E_{n \bmod 2 \wedge c} \quad \iota}{\forall E_{n \operatorname{div} 2}} \quad y}}{v}}{z}$$

where γ is an abbreviation for the term $\alpha_0(Add_0(\rho_0\, n, c)) = n + c$. We can now apply the equations of Table 3–16 to read off from this natural deduction tree the constraint term P. We do this in three steps corresponding to the three sub-terms marked P, P_3, P_2 in the above tree.

Evaluation of P_2:

$$\begin{aligned}
P_2 &\equiv \forall I_n \forall I_c \diamond F_y(\forall E_{n \bmod 2 \wedge c} \forall E_{n \operatorname{div} 2}\, z\ ,\ \iota\, y) \\
&\equiv \lambda n.\, \lambda c.\, \diamond F_y(\forall E_{n \bmod 2 \wedge c} \forall E_{n \operatorname{div} 2}\, z\ ,\ \iota\, y) \\
&\equiv \lambda n.\, \lambda c.\, (\pi_1(\forall E_{n \bmod 2 \wedge c} \forall E_{n \operatorname{div} 2}\, z)\ ,\ \iota\, \pi_2(\forall E_{n \bmod 2 \wedge c} \forall E_{n \operatorname{div} 2}\, z)) \\
&\equiv \lambda n.\, \lambda c.\, (\pi_1(z\,(n \operatorname{div} 2)\,(n \bmod 2 \wedge c))\ ,\ *)
\end{aligned}$$

The last equation holds because $\iota a = *$ irrespective of the shape of a. Note, P_2 is a term with free variable z.

Evaluation of P_3:

$$\begin{aligned}
P_3 &\equiv NatInd_{w,z}(\forall I_n \forall I_c \forall E_\gamma\, v\ ,\ P_2) \\
&\equiv NatInd_{w,z}(\lambda n.\, \lambda c.\, v\gamma\ ,\ P_2) \\
&\equiv natrec(\lambda n.\, \lambda c.\, v\gamma\ ,\ \lambda w.\, \lambda z.\, P_2) \\
&\equiv natrec(\lambda n.\, \lambda c.\, v\gamma\ ,\ \lambda w.\, \lambda z.\, \lambda n.\, \lambda c.\, (\pi_1(z\,(n \operatorname{div} 2)\,(n \bmod 2 \wedge c))\ ,\ *))
\end{aligned}$$

where $\gamma \equiv \alpha_0(Add_0(\rho_0\, n, c)) = n + c$. Note, P_3 is a term with free variable v. Variable z free in sub-term P_2 is bound by *natrec*. The recursion operator *natrec* which appears in the term P_3 reflects the induction used in the proof. It has type $natrec : \tau \Rightarrow (\mathbf{N} \Rightarrow \tau \Rightarrow \tau) \Rightarrow (\mathbf{N} \Rightarrow \tau)$ where τ is $\mathbf{N} \Rightarrow \mathbf{B} \Rightarrow (\Omega \times \mathbf{1})$.

Evaluation of P:

$$\begin{aligned}
P &\equiv Subst(\iota\,,\ \forall I_w \forall I_n\, Subst(\iota\,,\ \forall E_1 \forall E_n \forall E_w\, P_3)) \\
&\equiv \forall I_w \forall I_n \forall E_1 \forall E_n \forall E_w\, P_3 \\
&\equiv \lambda w.\, \lambda n.\, \forall E_1 \forall E_n \forall E_w\, P_3 \\
&\equiv \lambda w.\, \lambda n.\, P_3\, w\, n\, 1 \\
&\equiv \lambda w.\, \lambda n.\, (natrec(\lambda n.\, \lambda c.\, v\gamma\,, \\
&\qquad\qquad \lambda w.\, \lambda z.\, \lambda n.\, \lambda c.\, (\pi_1(z\,(n\ div\ 2)\,(n\ mod\ 2 \wedge c))\,,\ *)))\, w\, n\, 1
\end{aligned}$$

Note, P is a term of type $\mathbf{N} \Rightarrow \mathbf{N} \Rightarrow (\Omega^* \times \mathbf{1})$ with v as its only free variable.

We are interested in the proposition list $C(w,n) \equiv \pi_1(P\{?/v\}\, w\, n)$ which is thus completely determined by the structure of the correctness proof and the constraint γ introduced within this proof. But what does it say? It is far from obvious that this should be the expected overflow constraint. In fact, the lambda term obtained for $C(w,n)$ is not very illuminating. The term encodes an inductive process which can be rephrased as a recursive definition in the following way:

$$\begin{aligned}
C_0(0,n,c) &\stackrel{df}{=} ?\gamma \\
C_0(succ\, w, n, c) &\stackrel{df}{=} C_0(w, n\ div\ 2, n\ mod\ 2 \wedge c) \\
C(w,n) &\stackrel{df}{=} \pi_1 C_0(w,n,1)
\end{aligned}$$

Using this recursive definition we may try to give more familiar characterizations of $C(w,n)$ and $C_0(w,n,c)$. Two of such reformulations will be given below. The first is presenting $C_0(w,n,c)$ as an inequation at the abstract level and the second as an equation one part of which can be interpreted at the level of the bit-vector realization. Note, $C_0(w,n,c)$ is of type $\Omega^* \times \mathbf{1}$, so it always satisfies $\pi_2 C_0(w,n,c) = *$.

Proposition 4.1.2 *The proposition list* $\pi_1 C_0(w,n,c)$ *of type* Ω^* *has the form* $\pi_1 C_0(w,n,c) = [\phi]$ *such that* $\phi \cong n + c < 2^w$.

Proof: The proof is done in an informal way as opposed to a formal derivation in lax logic. There is, of course, no reason why it could not be formalized in the base logic, provided proofs of all arithmetical laws used are supplied as well. In particular the proof will bring into play the following two facts of Peano arithmetic:

$$\begin{aligned}(x+y)\,div\,2 &= x\,div\,2 + y\,div\,2 + (x\,mod\,2 \wedge y\,mod\,2)\\ x < 2\cdot y &\cong x\,div\,2 < y\end{aligned}$$

One proceeds by induction on w. For the base case $w = 0$ one finds that $\pi_1 C_0(0,n,c)$ is equal to $[\alpha_0(Add_0(\rho_0\, n, c)) = n + c]$ by definition of ? and γ, so it has the required form $[\phi]$. The left side of the equation $\phi \equiv \alpha_0(Add_0(\rho_0\, n, c)) = n + c$ can be simplified to 0 by the definition of α_0, Add_0, and ρ_0, so that ϕ is equivalent to $0 = n + c$, which in turn is in fact equivalent to condition $n + c < 2^0$. For the induction step one notes that $C_0(succ\, w, n, c) \equiv C_0(w, n\,div\,2, n\,mod\,2 \wedge c)$ and thus by induction hypothesis $\pi_1 C_0(succ\, w, n, c)$ must have the form $[\phi]$ with

$$\phi \cong n\,div\,2 + (n\,mod\,2 \wedge c) < 2^w.$$

But with the two laws above we obtain

$$\begin{aligned}\phi &\cong n\,div\,2 + (n\,mod\,2 \wedge c) < 2^w\\ &\cong n\,div\,2 + c\,div\,2 + (n\,mod\,2 \wedge c\,mod\,2) < 2^w\\ &\cong (n+c)\,div\,2 < 2^w\\ &\cong n + c < 2^{w+1}.\end{aligned}$$

The second equivalence is justified as c is an element of $\mathbf{B}$, or rather $\{0,1\}$, hence $c\,div\,2 = 0$ and $c\,mod\,2 = c$. ∎

Proposition 4.1.2 implies that the extracted constraint list $C(w,n) = \pi_1 C_0(w,n,1)$ modulo equivalence of propositions is in fact the expected overflow constraint

$n+1 < 2^w$, *i.e.* the condition on the abstract input n for which the dyadic representation of $succ\,n$ fits within w bits. Now, since $\Box[\phi] = \phi$ sequent 4.10 that we know to hold for $C(w,n)$, *viz.*

$$\vdash \forall w.\forall n.\,((\alpha_w \circ Inc_w \circ \rho_w)\,n = succ\,n)^{C(w,n)} \tag{4.11}$$

can now be reformulated as

$$\vdash \forall w.\forall n.\,n+1 < 2^w \supset (\alpha_w \circ Inc_w \circ \rho_w)\,n = succ\,n$$

which is precisely the approximation of the incrementor's specification that one would have had to guess if everything had to take place in the base logic alone. The important point here is that the constraint was constructed systematically in the course of a derivation in lax logic.

To finish off the incrementor, a second characterization of the constructed constraint will be given. A closer look suggests that the overflow constraint in general consist of two parts. Not only the result of the increment operation but also the input n has to be representable by w bits. It is a particular property of the incrementing operation that the latter follows from the former, so that only one condition remains. These two sources for the overflow constraint are made more explicit by the second characterization which is given next. As a more important difference it makes use of the possibility of indicating the incrementing overflow by the carry output of the low-level realization's most significant half-adder stage. Now, assuming such a carry-out signal for the w-bit incrementor is handed out as an extra signal $Carry_w(b,c)$, *i.e.*

$$\begin{aligned} Carry_0([\,],c) &\stackrel{df}{=} c \\ Carry_{w+1}(b_0 :: b,c) &\stackrel{df}{=} Carry_w(b, Carry(b_0,c)) \end{aligned}$$

the second characterization can be formulated:

Proposition 4.1.3 *The proposition list* $\pi_1 C_0(w,n,c)$ *of type* Ω^* *has the form* $\pi_1 C_0(w,n,c) = [\phi]$ *such that* $\phi \cong n\ div\ 2^w + Carry_w(\rho_w n, c) = 0$.

Proof: As before the proof is by induction on w and only given informally. The base case is straightforward since $\pi_1 C_0(0, n, c)$ is provably equal to $0 = n + c$ (*cf.* the proof of the previous proposition) and $n + c = n \; div \; 2^0 + Carry_0(\rho_0 n, c)$. The induction step is obtained as follows: We have

$$C_0(succ\, w, n, c) = C_0(w, n \; div \; 2, n \; mod \; 2 \wedge c)$$

and thus by induction hypothesis $\pi_1 C_0(succ\, w, n, c)$ is of form $[\phi]$ with

$$\phi \;\cong\; n \; div \; 2 \; div \; 2^w + Carry_w(\rho_w(n \; div \; 2), n \; mod \; 2 \wedge c) = 0.$$

From this we compute

$$\begin{aligned}
&(n \; div \; 2) \; div \; 2^w + Carry_w(\rho_w(n \; div \; 2), n \; mod \; 2 \wedge c) \\
&\quad = \; n \; div \; 2^{w+1} + Carry_w(\rho_w(n \; div \; 2)\,,\; Carry(n \; mod \; 2, c)) \\
&\quad = \; n \; div \; 2^{w+1} + Carry_{w+1}((n \; mod \; 2) :: \rho_w(n \; div \; 2)\,,\; c) \\
&\quad = \; n \; div \; 2^{w+1} + Carry_{w+1}(\rho_{w+1}\, n\,,\; c).
\end{aligned}$$

∎

Since for natural numbers the condition $x + y = 0$ is equivalent to $x = 0$ and $y = 0$ this second characterization directly leads to the constraint

$$n \; div \; 2^w = 0 \wedge Carry_w(\rho_w\, n, 1) = 0$$

The left part ensures that the input n fits within w bits and the right that there is no carry indication if the input is incremented as a bit-vector of length w. Note that if we put $w \equiv 1$, *i.e.* we are looking at a one-stage incrementor, this second condition becomes $Carry(\rho_1\, n, 1) = 0$, or equivalently, $\rho_1\, n \wedge 1 = 0$. This is precisely the condition for which the half-adder correctly adds input bits '$\rho_1\, n$' and '1', *i.e.* for which $Add(\rho_1\, n, 1) = \rho_1\, n + 1$.

* * *

The reader will notice (*cf.* Figure 4–4) that induction really is based at $w \equiv 0$ and not at $w \equiv 1$. In other words, verification is anchored in the 'zero'-bit incrementor rather than in the one-bit incrementor. Also, this is the only place where a constraint is explicitly introduced into the proof, *i.e.* where the hypothesis *let's-not-bother* is used. All the rest of the proof is merely manipulating and updating this constraint in order to get a constraint for arbitrary bit-length w and input n. Thus, the constraint really is induced by a constraint for the zero-bit case and the overflow constraint for the w-bit incrementor is nothing but the zero-bit case systematically pushed up through the induction. This is exactly the same situation as with the decrementor example. It is an open question if this is merely a peculiarity of these specific examples or a more generally occurring pattern, say for a certain class of abstractions.

4.1.4 Designing the Factorial

As the final example of this section consider the following design task involving both levels of data abstraction: At the abstract level the factorial is to be designed as a recursive function over natural numbers. The design is composed in the ideal world of natural number arithmetic using successor and multiplication as primitive operations[2]. These primitive operations will be assumed to be realized at a lower level as combinatory algorithms over finite bit-vectors. Since successor and multiplication (of natural numbers) can only be approximated on finite bit-vectors, the resulting realization of the factorial will only approximately be correct with respect to the initial abstract specification. The following will demonstrate how lax logic may be applied to keep track of and accumulate overflow constraints arising from the two subcomponents.

[2]The term 'primitive' means that we are not going to break down these operations themselves in terms of zero, successor, and primitive recursion.

Preliminaries

We begin by spelling out the design goal. We seek a function term $fac : \mathbf{N} \Rightarrow \mathbf{N}$ that satisfies the specification

$$\iota(fac\,0 = 1) \;\wedge\; \forall n^{\mathbf{N}}.\, \Diamond\iota(fac(succ\,n) = succ\,n \cdot fac\,n) \qquad (4.12)$$

where the $\Diamond$-operator indicates that the universal quantification may be relativized to some subset of natural numbers. A proof of this lax specification will eventually construct a constraint[3] $\gamma : \mathbf{N} \Rightarrow \Omega$, such that

$$fac\,0 = 1 \;\wedge\; \forall n^{\mathbf{N}}.\, \gamma\, n \;\supset\; fac(succ\,n) = succ\,n \cdot fac\,n$$

holds. For the design to be described it will be seen that γ characterizes the range of naturals for which no overflow occurs in the subcomponents. It will be useful to single out those parts of a specification that pertain to the base logic, so we write

$$\iota(F_0(fac)) \;\wedge\; \forall n^{\mathbf{N}}.\, \Diamond\iota(F_1(fac))$$

for the factorial's specification, where $F_0(f) \stackrel{df}{\equiv} f\,0 = 1$ and $F_1(f) \stackrel{df}{\equiv} f(succ\,n) = succ\,n \cdot f\,n$.

The design will go through three phases, called *Modularization*, *Realization*, and *Composition*, which we briefly outline below.

MODULARIZATION. The factorial $fac = a\text{-}comp(inc, mul)$ is decomposed into an incrementor $inc : \mathbf{N} \Rightarrow \mathbf{N}$ and a multiplier $mul : \mathbf{N} \times \mathbf{N} \Rightarrow \mathbf{N}$ with subspecifications

$$\begin{array}{lll} \forall n^{\mathbf{N}}.\, \Diamond\iota I(inc) & I(inc) & \stackrel{df}{\equiv}\; inc\,n = succ\,n \\ \forall n_1^{\mathbf{N}}.\, \forall n_2^{\mathbf{N}}.\, \Diamond\iota M(mul) & M(mul) & \stackrel{df}{\equiv}\; mul(n_1, n_2) = n_1 \cdot n_2. \end{array}$$

[3] It will actually be a function of type $1 \times (\mathbf{N} \Rightarrow (\Omega^* \times 1))$.

It will be verified that given implementations for each of the two sub-specifications, then their composition satisfies the specification of the factorial, *i.e.*

$$\forall n.\, \Diamond\iota I(i)\,,\ \forall n_1.\forall n_2.\, \Diamond\iota M(m) \ \vdash_{i,m} \ \iota(F_0(a\text{-}comp(i,m))) \wedge \forall n^{\mathbf{N}}.\, \Diamond\iota(F_1(a\text{-}comp(i,m))).$$

The composition operator *a-comp* essentially will be the iteration operator for natural numbers. Consequently, the central proof technique will be natural induction. Note, specifications for incrementor and multiplier, too, are lax propositions, *i.e.* they contain $\Diamond$ in order to account for overflow constraints.

REALIZATION. The subcomponents *inc* and *mul* need to be realized as operations $Inc_w : \mathbf{B}^w \Rightarrow \mathbf{B}^w$ and $Mul_w : \mathbf{B}^w \times \mathbf{B}^w \Rightarrow \mathbf{B}^w$ over w bits, such that their abstractions satisfy the postulated abstract specifications. The verification goals are[4]

$$\forall w.\, \forall n.\, \Diamond\iota I(\alpha_w \circ Inc_w \circ \rho_w) \qquad \forall w.\, \forall n_1.\, \forall n_2.\, \Diamond\iota M(\alpha_w \circ Mul_w \circ (\rho_w \times \rho_w))$$

where $\times$ stands for the product combinator $f \times g = \lambda x.\,(f(\pi_1 x), g(\pi_2 x))$. The realization of the w-bit incrementor Inc_w has been given and proven correct in Section 4.1.3 already. Using this Inc_w and its correctness proof will introduce a specific overflow constraint into the design of the factorial. As to the other subcomponent there are many different ways of implementing a w-bit multiplier, sequential as well as combinational. A detailed verification would be more complicated than for the case of the incrementor while it is essentially of the same flavour. As it would probably not give new insight into the ways constraints emerge and accumulate in a proof of lax logic we will not work through a specific realization, Mul_w, of the multiplier. Instead some reasonable assumptions will be made about the correctness proof as well as the constraint that it carries, and the design of the factorial will be completed from there.

[4]Since Mul_w is a function of two arguments we need to use the realization mapping twice, *i.e.* we have to precompose it with $\rho_w \times \rho_w$.

As the result of the first two phases one obtains two low-level subcomponents Inc_w, Mul_w and a proof that if their behaviour is *first* abstracted and *then* composed (at the abstract level) one obtains a behaviour which, under certain constraints, satisfies the specification of the factorial. More formally, one concludes

$$\vdash \forall w.\ \iota(F_0(fac_w)) \ \wedge\ \forall n^{\mathbf{N}}.\Diamond\iota(F_1(fac_w))$$

where

$$fac_w \stackrel{df}{\equiv} a\text{-}comp(\alpha_w \circ Inc_w \circ \rho_w\ ,\ \alpha_w \circ Mul_w \circ (\rho_w \times \rho_w)).$$

Although the design may be considered completed now, for a function (in fact, a family of functions) has been constructed that satisfies the factorial's specification, this particular function may not be what one is ultimately looking for. One might prefer *first* to compose the subcomponents as low-level bit-operations and *then* abstract the resulting composite behaviour to yield a realization of the factorial. This gets us to yet another, third design phase:

COMPOSITION. The task is to decide for a suitable low-level composition operator $r\text{-}comp_w$ and to prove

$$\vdash \forall w.\ \iota(F_0(\alpha_w \circ Fac_w \circ \rho_w)) \ \wedge\ \forall n^{\mathbf{N}}.\Diamond\iota(F_1(\alpha_w \circ Fac_w \circ \rho_w))$$

where

$$Fac_w \stackrel{df}{\equiv} r\text{-}comp_w(Inc_w, Mul_w).$$

Fac_w, then, is a correct low-level realization of the factorial. The operator $r\text{-}comp_w$ might be a direct realization at the lower level of *a-comp* from the abstract level, such that

$$\forall f, g.\ a\text{-}comp(\alpha_w \circ f \circ \rho_w\ ,\ \alpha_w \circ g \circ (\rho_w \times \rho_w)) = \alpha_w \circ r\text{-}comp_w(f, g) \circ \rho_w.$$

Although important, this third phase will be discussed only very briefly.

Decomposing the Factorial

The goal in the modularization phase is to implement the factorial from subcomponents at the abstract level. If the $\Diamond$ and ι operators are ignored (4.12) is turned into an ordinary equational specification which may be read as a recursive definition for *fac*:

$$\begin{aligned} \mathit{fun}\ \ \mathit{fac}\,0 &= 1 \\ \mid\ \mathit{fac}(\mathit{succ}\,n) &= \mathit{succ}\,n \cdot \mathit{fac}\,n. \end{aligned}$$

From this one directly obtains a solution for *fac* in terms of iteration *iter* and implementations for successor and multiplication. To this end we anticipate that a solution for a recursive definition of form[5]

$$\begin{aligned} \mathit{fun}\ \ f\,0 &= a \\ \mid\ f(\mathit{succ}\,n) &= h(\mathit{succ}\,n, f\,n) \end{aligned}$$

is obtained from iteration by

$$f \equiv \pi_2(\mathit{iter}_x\,((1,a),(\mathit{succ}(\pi_1\,x),h\,x))).$$

Now suppose there are functions $\mathit{inc} : \mathbf{N} \Rightarrow \mathbf{N}$ and $\mathit{mul} : (\mathbf{N} \times \mathbf{N}) \Rightarrow \mathbf{N}$ available which are *approximations* of the successor and multiplication operation in that they satisfy

$$\forall n.\, \Diamond\iota I(\mathit{inc}) \qquad \forall n_1.\, \forall n_2.\, \Diamond\iota M(\mathit{mul}).$$

Trivial solutions, of course, are $\mathit{inc} = \lambda n.\, \mathit{succ}\,n$ and $\mathit{mul} = \lambda x.\, (\pi_1 x) \cdot (\pi_2 x)$. In view of what was said above we are lead to put

$$\mathit{a\text{-}comp}(\mathit{inc}, \mathit{mul}) \stackrel{df}{\equiv} \pi_2(\mathit{iter}_x\,((1,1),(\mathit{inc}(\pi_1\,x), \mathit{mul}\,x))). \qquad (4.13)$$

as the composition operator. The goal for verifying that this is in fact a correct decomposition of the factorial becomes

$$\forall n.\, \Diamond\iota I(i)\ ,\ \forall n_1.\, \forall n_2.\, \Diamond\iota M(m)\ \vdash_{i,m}\ \iota(F_0(\mathit{a\text{-}comp}(i,m))) \qquad (4.14)$$
$$\wedge\, \forall n^{\mathbf{N}}.\, \Diamond\iota(F_1(\mathit{a\text{-}comp}(i,m)))$$

[5]This scheme is not general primitive recursion. It is a special version that admits a specialized solution.

where i, m are variables of proper types. Note that it is only for the fact that *inc* and *mul* are going to be approximations of successor and multiplication that lax logic is needed at all. Otherwise the result would immediately follow from the general solution of the recursive definitions mentioned above. In fact, this standard proof basically will be reconstructed in lax logic.

We are now going to prove (4.14) in lax logic. First we introduce two useful abbreviations:

$$\begin{aligned} fac &\stackrel{df}{\equiv} a\text{-}comp(i, m) \\ cnt &\stackrel{df}{\equiv} \pi_1(iter_x\,((1,1),(i(\pi_1\,x), m\,x))) \end{aligned}$$

Derivation 2 *If incrementor* $inc : \mathbf{N} \Rightarrow \mathbf{N}$ *satisfies the specification* $\forall n.\, \Diamond\iota I(inc)$, *then so does* $cnt\{inc/i\}\{mul/m\} : \mathbf{N} \Rightarrow \mathbf{N}$, *where* $mul : \mathbf{N} \times \mathbf{N} \Rightarrow \mathbf{N}$ *arbitrary, i.e.* $\forall n.\, \Diamond\iota I(i) \vdash_{i,m} \forall n.\, \Diamond\iota I(cnt)$.

The proof tree is shown in Figure 4–7. It refers implicitly (rules ι) to two sub-derivations in the base logic, *viz.* of the equations

$$cnt\,0 = succ\,0 \quad \text{and} \quad cnt(succ\,n) = i(cnt\,n)$$

which are easily obtained exploiting standard laws for products and recursion equations for *iter*. Just like before these computations do not contribute to the constraint term and therefore can be omitted. The proof employs standard induction over natural numbers. No special care needs to be taken to account for the $\Diamond$-operator. The presence of $\Diamond$ only influences the proof in a very trivial way: Removing all instances of $\Diamond$, ι and the application of related rules $\Diamond I$, $\Diamond L$, ι results in a well-formed proof in the base logic, and we claim there is a canonical process turning that ordinary proof back into the one in Figure 4–7. In particular, there is no non-canonical introduction of constraints. Recall that rule $\Diamond I$ introduces the vacuous constraint $[\,]$.

With Derivation 2 at hand we can demonstrate the modularization proof for (4.14) as in Figure 4–8. It does not involve induction and refers to two equational

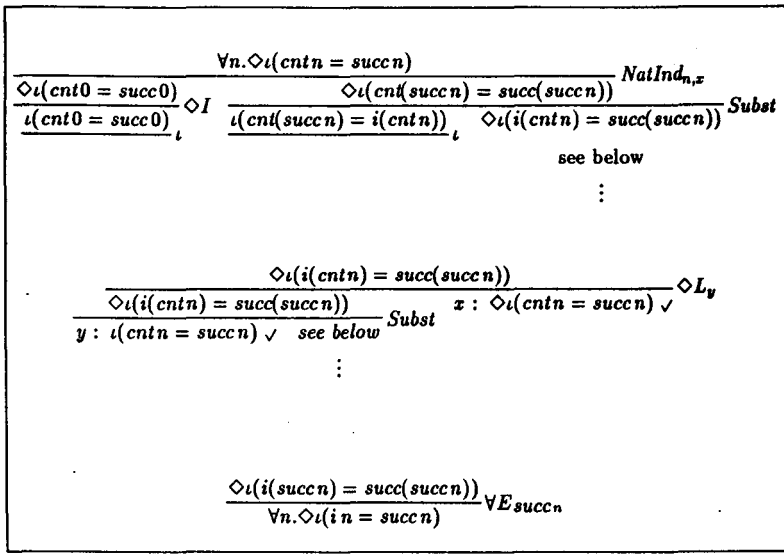


Figure 4–7: Proof of Derivation 2

sub-derivations in the base logic which again are not difficult to obtain. Thus, we can set down

Derivation 3 *Given an incrementor* $inc : \mathbf{N} \Rightarrow \mathbf{N}$ *and a multiplier* $mul : \mathbf{N} \times \mathbf{N} \Rightarrow \mathbf{N}$ *which satisfy their abstract specifications, then their composition* $a\text{-}comp(inc, mul)$ *satisfies the specification of the factorial, i.e.*

$$\forall n.\, \Diamond\iota I(i)\ ,\ \forall n_1.\forall n_2.\Diamond\iota M(m)$$
$$\vdash_{i,m} \iota F_0(a\text{-}comp(i,m)) \wedge \forall n^{\mathbf{N}}.\Diamond\iota F_1(a\text{-}comp(i,m)).$$

Constraint Analysis for Decomposition

We may now be interested in the constraint handling potential of Derivation 3:

$$\forall n.\, \Diamond\iota I(i)\ ,\ \forall n_1.\forall n_2.\Diamond\iota M(m) \tag{4.15}$$
$$\vdash_{i,m} \iota F_0(a\text{-}comp(i,m)) \wedge \forall n^{\mathbf{N}}.\Diamond\iota F_1(a\text{-}comp(i,m))$$

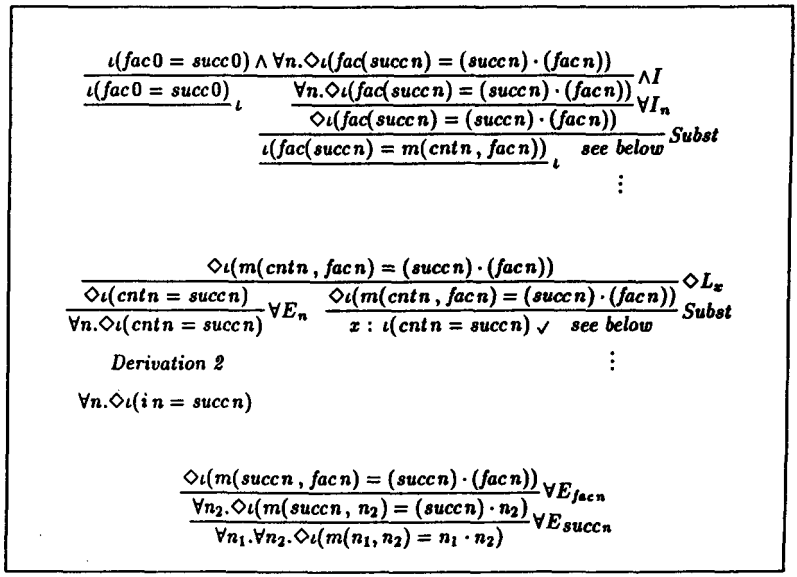


Figure 4–8: Proof of Derivation 3

Remember that every ◇ in a lax formula stands for a constraint, or rather a constraint list of type Ω^* by which the formula is to be weakened at this point. The constraint itself is determined by a proof of the formula. In this spirit, the derivation of (4.15) determines a function that transforms a constraint of the incrementor (represented by ◇ in the first hypothesis) and a constraint of the multiplier (represented by ◇ in the second hypothesis) into a constraint for the factorial (represented by ◇ in the assertion). Constraint extraction allows us to analyze this constraint transforming function. The constraint types of hypotheses and assertion are

$$
\begin{aligned}
|\forall n.\Diamond\iota I(i)| &= \mathbf{N} \Rightarrow (\Omega^* \times \mathbf{1}) \\
|\forall n_1.\forall n_2.\Diamond\iota M(m)| &= \mathbf{N} \Rightarrow \mathbf{N} \Rightarrow (\Omega^* \times \mathbf{1}) \\
|\iota F_0(a\text{-}comp(i,m)) \wedge \forall n.\Diamond\iota F_1(a\text{-}comp(i,m))| &= \mathbf{1} \times (\mathbf{N} \Rightarrow (\Omega^* \times \mathbf{1}))
\end{aligned}
$$

Given a variable u of type $\mathbf{N} \Rightarrow (\Omega^* \times 1)$ and a variable v of type $\mathbf{N} \Rightarrow \mathbf{N} \Rightarrow (\Omega^* \times 1)$ we will extract by the translation process defined in Section 3.2 a constraint term R of type $1 \times (\mathbf{N} \Rightarrow (\Omega^* \times 1))$ with free variables i, m, u, v, such that

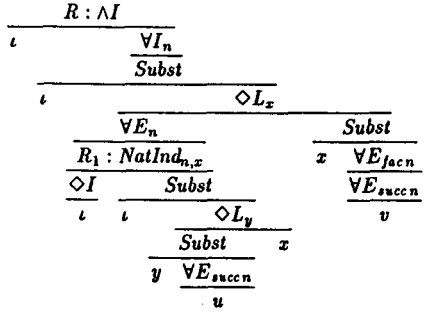
$$(\forall n.\, \Diamond_\iota I(i))^\# u\ ,\ (\forall n_1.\, \forall n_2.\, \Diamond_\iota M(m))^\# v$$
$$\vdash_{i,m,u,v} (\iota F_0(a\text{-}comp(i,m)) \wedge \forall n^{\mathbf{N}}.\, \Diamond_\iota F_1(a\text{-}comp(i,m)))^\# R$$

or, after expanding the definition of constraint predicates

$$\forall n.\, I(i)^{\pi_1(u\,n)}\ ,\ \forall n_1.\, \forall n_2.\, M(m)^{\pi_1(v\,n_1\,n_2)} \tag{4.16}$$
$$\vdash_{i,m,u,v} F_0(a\text{-}comp(i,m)) \wedge \forall n.\, F_1(a\text{-}comp(i,m))^{\pi_1((\pi_2 R)\,n)}$$

In the sequel the term $\pi_1((\pi_2 R)\, n)$ will be computed and interpreted.

The computation of R works from the natural deduction proof trees of Figures 4–7 and 4–8 of sequent (4.15). Both trees plugged together and reduced to rule applications are shown below



The point where we break up this tree into two digestible pieces is indicated with R_1.

Evaluation of R_1:

$$R_1 \equiv NatInd_{n,x}(\Diamond I\, \iota\,,\ Subst(\iota\,,\ \Diamond L_y(Subst(y, \forall E_{succ\,n}\, u)\,,\ x)))$$
$$\equiv NatInd_{n,x}(\Diamond I\, \iota\,,\ \Diamond L_y(\forall E_{succ\,n}\, u\,,\ x))$$

$$\equiv \mathit{NatInd}_{n,x}(\Diamond I *\,,\, \Diamond L_y(\forall E_{succ\,n}\, u\,,\, x))$$
$$\equiv \mathit{NatInd}_{n,x}(([\,],*)\,,\, \Diamond L_y(u\,(succ\,n)\,,\, x))$$
$$\equiv \mathit{NatInd}_{n,x}(([\,],*)\,,\, (\pi_1(u\,(succ\,n))\,@\,\pi_1 x\,,\, \pi_2(u\,(succ\,n))))$$
$$\equiv \mathit{natrec}(([\,],*)\,,\, \lambda n.\,\lambda x.\,(\pi_1(u\,(succ\,n))\,@\,\pi_1 x\,,\, \pi_2(u\,(succ\,n))))$$

This is what we get for R_1 by directly translating the proof tree. Note, R_1 has u as its only free variable. Before we continue to compute R, *i.e.* the constraint term for the whole tree, R_1 will be simplified a bit. Since variable u is of type $\mathbf{N} \Rightarrow (\Omega^* \times \mathbf{1})$, the sub-term $\pi_2(u\,(succ\,n))$ is of type $\mathbf{1}$ and hence we can prove $\pi_2(u\,(succ\,n)) = *$. This, for a start, gives

$$R_1 \;=\; \mathit{natrec}(([\,],*)\,,\, \lambda n.\,\lambda x.\,(\pi_1(u\,(succ\,n))\,@\,\pi_1 x\,,\, *)).$$

Further, the recursion combinator *natrec* has type $(\alpha \times (\mathbf{N} \Rightarrow \alpha \Rightarrow \alpha)) \Rightarrow \mathbf{N} \Rightarrow \alpha$ with, in this case, $\alpha = \Omega^* \times \mathbf{1}$. The component of type $\mathbf{1}$ is redundant so that the recursion can further be simplified by removing the construction of $*$ which is merely carried through the recursion as a dummy. Exploiting the canonical isomorphism

$$\Omega^* \times \mathbf{1} \;\cong\; \Omega^*$$

it can be proven by induction that

$$\pi_1(R_1\,n) \;=\; \mathit{natrec}([\,]\,,\, \lambda n.\,\lambda x.\,\pi_1(u\,(succ\,n))\,@\,x)\,n$$

where now $\alpha = \Omega^*$ and x has type Ω^* (in contrast to $\Omega^* \times \mathbf{1}$ before). Note, $\pi_2(R_1\,n)$ has type $\mathbf{1}$ whence $\pi_2(R_1\,n) = *$. One immediately obtains an intuitive picture of what is going on when $\pi_1(R_1\,n)$ is evaluated for the first few n:

$$\begin{aligned}
\pi_1(R_1\,0) &= [\,] \\
\pi_1(R_1\,(succ\,0)) &= \pi_1(u\,(succ\,0))\,@\,[\,] \\
\pi_1(R_1\,(succ\,(succ\,0)) &= \pi_1(u\,(succ\,(succ\,0)))\,@\,\pi_1(u\,(succ\,0))\,@\,[\,] \\
&\vdots
\end{aligned}$$

Thus, for any given natural number n, the constraint list $\pi_1(R_1\, n)$ is obtained by accumulating the constraint lists $u\, k$ with $k = 1, \ldots, n$ into a single list.

So much for the analysis of R_1. We will be able to simplify $\pi_1(R_1\, n)$ further when the concrete input constraint of the incrementor will be plugged in for variable u. (Remember that u stands for a proof of the formula $\forall n.\, \Diamond \iota I(i)$, the specification of the incrementor.)

Evaluation of R:

$$\begin{aligned} R &\equiv \wedge I(\iota\,, \forall I_n\, \mathit{Subst}(\iota\,, \Diamond L_x(\forall E_n\, R_1\,, \mathit{Subst}(x, \forall E_{\mathit{fac}\, n}\, \forall E_{\mathit{succ}\, n}\, v)))) \\ &\equiv \wedge I(\iota\,, \forall I_n\, \Diamond L_x(\forall E_n\, R_1\,, \forall E_{\mathit{fac}\, n}\, \forall E_{\mathit{succ}\, n}\, v)) \\ &\equiv \wedge I(*\,, \forall I_n\, \Diamond L_x(R_1\, n\,, v\,(\mathit{succ}\, n)\,(\mathit{fac}\, n))) \\ &\equiv (*\,, \lambda n.\, (\pi_1(R_1\, n) \mathbin{@} \pi_1(v\,(\mathit{succ}\, n)\,(\mathit{fac}\, n))\,, \pi_2(R_1\, n))) \\ &= (*\,, \lambda n.\, (\pi_1(R_1\, n) \mathbin{@} \pi_1(v\,(\mathit{succ}\, n)\,(\mathit{fac}\, n))\,, *)) \end{aligned}$$

R has free variables u, v, i, m, the latter two because of $\mathit{fac} \equiv \mathit{a\text{-}comp}(i, m)$.

Here, we can return to the point of departure, *viz.* the sequent (4.16). It says that for any input constraint $u : \mathrm{N} \Rightarrow (\Omega^* \times 1)$ for the incrementor and input constraint $v : \mathrm{N} \Rightarrow \mathrm{N} \Rightarrow (\Omega^* \times 1)$ for the multiplier, $\pi_1((\pi_2 R)\, n) : \Omega^*$ defines a (sufficient) input constraint for the factorial at input n. Now, the term $\pi_1((\pi_2 R)\, n)$ can be seen to satisfy

$$\pi_1((\pi_2 R)\, n) \;=\; \pi_1(R_1\, n) \mathbin{@} \pi_1(v\,(\mathit{succ}\, n)\,(\mathit{fac}\, n))$$

so that we get a direct interpretation of the factorial's input constraint in terms of those for incrementor and multiplier:

> *For a given input n of the factorial, the constraint list for n contains all the constraints of the incrementor for inputs $k = 1, \ldots, n$ (cf. the analysis of $\pi_1(R_1\, n)$ above) and the input constraint for the multiplier at input pair $(\mathit{succ}\, n\,, \mathit{fac}\, n)$.*

In particular, if both subcomponents are an ideal incrementor and an ideal multiplier, as it is the case for $\mathit{inc} \equiv \lambda n.\, \mathit{succ}\, n$ and $\mathit{mul} \equiv \lambda x.\, (\pi_1 x) \cdot (\pi_2 x)$, then

since the constraint lists $\pi_1(u\,n)$ and $\pi_1(v\,n\,m)$ are empty[6] the constraint list $\pi_1((\pi_2 R)\,n)$ for the factorial is empty, too.

Realization

After decomposing the factorial and its specification the next design step consists of realizing the subcomponents at the lower abstraction level as bit-operations. For the incrementor this has been done already in Section 4.1.3. There we have constructed a constraint term $C(w,n)$ of type Ω^* depending on variables w and n, where w is the bit-length and n the input to the incrementor, such that by (4.11)

$$\vdash \forall w.\forall n.\ I(\alpha_w \circ Inc_w \circ \rho_w)^{C(w,n)}. \tag{4.17}$$

It was shown that $C(w,n)$ is always a list with one element, $C(w,n) = [\phi]$, and that ϕ is equivalent to the condition $n + c < 2^w$ (*cf.* Proposition 4.1.2) and $n\ div\ 2^w + Carry_w(\rho_w n, c) = 0$ (*cf.* Proposition 4.1.3).

To complete the task of realizing the factorial we ought to deal with the multiplier. However, because of the similarity to the problem of realizing the incrementor this does not appear to provide any new insights while it is likely to be an unreasonably excessive exercise in applying the formalism of lax logic. Therefore, the multiplier will be left to a future software implementation of lax logic. Of course, the interesting question will be if there one also can do away with constraints in a similarly canonical way as in the case of the incrementor and decrementor. Here, it will be simply assumed that there is some function term $Mul_w : \mathbf{B}^w \times \mathbf{B}^w \Rightarrow \mathbf{B}^w$ and constraint list $D(w,n_1,n_2) : \Omega^*$ such that

$$\vdash \forall w.\forall n_1.\forall n_2.\ M(\alpha_w \circ Mul_w \circ (\rho_w \times \rho_w))^{D(w,n_1,n_2)}. \tag{4.18}$$

With the proposition list $D(w,n_1,n_2)$ being the overflow constraint of the w-bit multiplier it can probably be shown analogously to the incrementor that $D(w,n_1,n_2) = [\phi]$ such that $\phi \cong n_1 \cdot n_2 < 2^w$.

[6] Of course, this ought to be verified.

Now that the decomposition of the factorial and the realization of its subcomponents has been completed we turn to analyzing the achievement in terms of the constraints accumulated within the obtained correctness proofs. Fix a particular bit-length $w : \mathbf{N}$. With realizations of both subcomponents at hand we can now proceed bottom-up and derive the overflow constraint induced if w-bit incrementor and multiplier are composed to give a w-bit realization of the factorial

$$fac_w : \mathbf{N} \to \mathbf{N} \quad \stackrel{df}{=} \quad a\text{-}comp(\alpha_w \circ Inc_w \circ \rho_w\,,\, \alpha_w \circ Mul_w \circ (\rho_w \times \rho_w)).$$

Specializing variables u, v, i, m in sequent (4.16) as follows

$$\begin{aligned} u &:= \lambda n.\,(C(w,n)\,,\,*) \\ v &:= \lambda n_1.\,\lambda n_2.\,(D(w,n_1,n_2)\,,\,*) \\ i &:= \alpha_w \circ Inc_w \circ \rho_w \\ m &:= \alpha_w \circ Mul_w \circ (\rho_w \times \rho_w) \end{aligned}$$

and a little equality reasoning involving β-reduction for products and pairs yields the sequent

$$\begin{aligned} &\forall w.\,\forall n.\; I(\alpha_w \circ Inc_w \circ \rho_w)^{C(w,n)}\,, \\ &\quad \forall w.\,\forall n_1.\,\forall n_2.\; M(\alpha_w \circ Mul_w \circ (\rho_w \times \rho_w))^{D(w,n_1,n_2)} \\ &\qquad \vdash_w F_0(fac_w) \;\wedge\; \forall n.\; F_1(fac_w)^{E(w,n)} \end{aligned} \tag{4.19}$$

where

$$\begin{aligned} E(w,n) \quad \stackrel{df}{=} \quad & \pi_1((\pi_2 R)\,n) \\ & \{\lambda n.\,(C(w,n)\,,\,*)/u\}\{\lambda n_1.\,\lambda n_2.\,(D(w,n_1,n_2)\,,\,*)/v\} \\ & \{\alpha_w \circ Inc_w \circ \rho_w/i\}\{\alpha_w \circ Mul_w \circ (\rho_w \times \rho_w)/m\}. \end{aligned}$$

Sequent (4.19) contains the information on how the constraints of incrementor and multiplier are composed to yield the constraint for the factorial. Namely, if Sequent (4.19) is plugged together with sequents (4.17) and (4.18) one obtains

$$\vdash_w \quad F_0(fac_w) \;\wedge\; \forall n.\; F_1(fac_w)^{E(w,n)} \tag{4.20}$$

which proves the correctness of the w-bit factorial under constraint $E(w,n)$ which really is a list of propositions. We can use the function $\sqcap : \Omega^* \Rightarrow \Omega$ that *conjoins* ($\wedge$) a list of propositions into a single proposition to transform sequent (4.20) into the equivalent

$$\vdash_w \quad F_0(fac_w) \wedge \forall n.\, \sqcap E(w,n) \supset F_1(fac_w)$$

which says that the proposition $\sqcap E(w,n)$ is a sufficient condition on the input for which the w-bit implementation fac_w of the factorial satisfies its abstract specification. It is remarked again, that $E(w,n)$ is systematically extracted from the realization proofs for incrementor and multiplier and the decomposition proof of the factorial. In the following we want to obtain an intuitive characterization of the input constraint $\sqcap E(w,n)$.

With the abbreviation $R'_1 =_{df} (\pi_1(R_1\, n))\{\lambda n.\,(C(w,n)\,,\, *)/u\}$ we compute

$$E(w,n) \;\equiv\; R'_1 \mathbin{@} D(w, succ\, n, fac_w\, n).$$

Hence, $\sqcap E(w,n) = (\sqcap R'_1) \wedge D(w, succ\, n, fac_w\, n)$ by definition of $\sqcap$. So, we are left with the job of working out the condition $\sqcap R'_1$. To this end let us refine the comments above about list $\pi_1(R_1\, n)$ to see what the proposition list R'_1 looks like. First, we have

$$\begin{aligned} R'_1 &\equiv (\pi_1(R_1\, n))\{\lambda n.\,(C(w,n)\,,\, *)/u\} \\ &\equiv natrec([\,], \lambda n.\, \lambda x.\, \pi_1((\lambda n.\,(C(w,n)\,,\, *))\,(succ\, n)) \mathbin{@} x)\, n \\ &= natrec([\,], \lambda n.\, \lambda x.\, C(w, succ\, n) \mathbin{@} x)\, n. \end{aligned}$$

So, evaluating R'_1 for the first few n makes clear what is going on:

$$\begin{aligned} R'_1\{0/n\} &= [\,] \\ R'_1\{succ\, 0/n\} &= C(w, succ\, 0) \mathbin{@} [\,] \\ R'_1\{succ\,(succ\, 0)/n\} &= C(w, succ\,(succ\, 0)) \mathbin{@} C(w, succ\, 0) \mathbin{@} [\,] \\ &\vdots \end{aligned}$$

For every $n > 0$ the list R'_1 contains all propositions $C(w,k)$ for $0 < k \leq n$. For $n \equiv 0$ the list is empty. Thus,

$$\sqcap R'_1 \;\cong\; \forall 0 < k \leq n.\, C(w,k)$$

which is verified formally by an induction argument. This can be simplified even further bearing in mind that in our particular case $C(w,n) \cong n+1 < 2^w$ which exhibits the property that for all n, $C(w, succ\, n)$ implies $C(w,n)$ which means

$$\forall 0 < k \leq n.\, C(w,k) \;\cong\; n \geq 1 \supset C(w,n).$$

Thus we find, modulo equivalence of propositions, that $\sqcap R'_1$ is the same as the condition $n \geq 1 \supset C(w,n)$, and we arrive at the final result

Derivation 4 *The w-bit realization of the factorial*

$$fac_w = a\text{-}comp(\alpha_w \circ Inc_w \circ \rho_w\,,\; \alpha_w \circ Mul_w \circ (\rho_w \times \rho_w))$$

satisfies specification

$$fac_w\, 0 = 0 \;\wedge\; \forall n.\, \gamma_w\, n \supset fac_w(succ\, n) = (succ\, n) \cdot fac_w\, n$$

where $\gamma_w\, n$ *is the condition*

$$\gamma_w\, n \;\stackrel{df}{\equiv}\; (succ\, n) \cdot (fac_w\, n) < 2^w \;\wedge\; n \geq 1 \supset n+1 < 2^w.$$

The first half of $\gamma_w\, n$, the condition $(succ\, n) \cdot (fac_w\, n) < 2^w$, stems from the constraint $D(w, succ\, n, fac_w\, n)$ (*cf.* the assumptions made above about the multiplier constraint).

* * *

Summing up, the essential point of the example is that by using lax logic the overflow constraint has been systematically accumulated in the verification of the factorial and that this constraint can be analyzed in a completely independent step *after* (lax) correctness is established. Further, no fancy induction principle involving the $\Diamond$-operator was necessary, ordinary induction over natural numbers proved to be adequate.

Concrete-Level Composition

Strictly speaking the goal of designing the abstract factorial from bit-level operations may be considered achieved at this point. It was shown that the function term

$$fac_w = a\text{-}comp(\alpha_w \circ Inc_w \circ \rho_w\,,\ \alpha_w \circ Mul_w \circ (\rho_w \times \rho_w))$$

satisfies the specification of the factorial. But as mentioned in the beginning fac_w might not be exactly what one is looking for. It may be more appropriate to ask for a concrete-level composition operator $r\text{-}comp_w$ such that the abstraction of the concrete-level function

$$Fac_w : \mathbf{B}^w \Rightarrow \mathbf{B}^w \stackrel{df}{\equiv} r\text{-}comp_w(Inc_w, Mul_w)$$

satisfies the specification of the factorial, *i.e.* one has

$$\vdash_w \ F_0(\alpha_w \circ Fac_w \circ \rho_w) \wedge \forall n.\, \Diamond\iota F_1(\alpha_w \circ Fac_w \circ \rho_w).$$

There will be many ways to define such an operator $r\text{-}comp_w$ each of which is encapsulating a different design style. If, for instance, Fac_w is to be interpreted as a piece of hardware and Inc_w, Mul_w as combinatory circuits, then the options range from combinatory to sequential composition and in the latter case further from synchronous to asynchronous computation. All this is involved enough a story to be worth being discussed all by itself and beyond the scope of this thesis.

Once $r\text{-}comp_w$ is defined one would proceed by showing that $r\text{-}comp_w$ is a correct realization of the abstract composition operator *a-comp* in the sense that

$$\forall f, g.\ \alpha_w \circ r\text{-}comp_w(f,g) \circ \rho_w = a\text{-}comp(\alpha_w \circ f \circ \rho_w\,,\ \alpha_w \circ g \circ (\rho_w \times \rho_w))$$

which immediately would allow us to deduce $F(\alpha_w \circ Fac_w \circ \rho_w)$ from $F(fac_w)$ no matter what property F is by simple equality substitution. This is the ideal world. However, just as Inc_w and Mul_w are approximative realizations of the abstract incrementor and multiplier the operator $r\text{-}comp_w$ may be a realization of *a-comp*

only *up to* certain constraints. If, for instance Inc_w and Mul_w were synchronous sequential circuits, then the low-level composition results in a correct synchronous system only under a constraint on the length of the clock phase depending on delay parameters in both sub-circuits. How these constraints can be accounted for by using lax logic and by introducing $\Diamond$ in the right places is a question that will have to be answered by future research.

4.2 Example of Synchronous Hardware Design

In this section we take up again the motivating example from Section 2.3 in designing a synchronous *modulo*-2 counter. This example differs from the previous ones fundamentally in that it employs a relational rather than functional approach for modelling the behaviour of components. Its main purpose, besides taking a hardware application, is to demonstrate the use of lax logic on a different description paradigm.

To keep explanations reasonably short we will not mention all details that would have to be spelled out in a completely formal presentation. Among those are for instance all definitions that have to do with basic data types such as integers and booleans. We simply assume that these data types along with their usual mathematical properties are available in the base logic. We may thus focus on those parts of the verification that are done within $\mathcal{L}$ proper.

The task, set up in Section 2.3.2, is to find a derivation for

$$cnt\_abs\,(x, y, z) \vdash_{x,y} cnt\_appr\,(x, y) \tag{4.21}$$

where here and in the sequel x, y, z are distinct variables of type ***signal***, and

$$\begin{aligned}
cnt_abs\,(x,y,z) &= xor_abs\,(x,z,y) \wedge latch_abs\,(y,z)\\
cnt_appr\,(x,y) &= (C_0 \wedge C_1(x,y)) \supset (\forall t_1,t_2.\, C_2(t_1,t_2)\\
&\qquad \supset (next_abs\,(t_1,t_2) \supset y(t_2) = x(t_2) + y(t_1)))\\
xor_abs\,(x,y,z) &= (stable\,x\,\delta_X \wedge stable\,y\,\delta_X)\\
&\qquad \supset \forall t.\, tick\ t \supset zt = xt + yt\\
latch_abs\,(d,q) &= (one_shot \wedge min_sep\,\delta_L) \supset\\
&\qquad \supset \forall t_1,t_2.(tick\ t_1 \wedge tick\ t_2) \supset\\
&\qquad\quad (next_abs(t_1\,,\,t_2) \supset q(t_2) = d(t_1))\\
next_abs(t_1\,,\,t_2) &= t_1 < t_2 \wedge \forall t.\, tick\ t \supset (t_1 < t \supset t_2 \leq t)
\end{aligned}$$

where here and in the following t, t_1, t_2 are distinct variables of type ***int***. The derivation of (4.21) is to be split into a main part that is free of constraints, and a successive constraint analysis to establish constraints C_0, C_1, C_2 for the composite device. The first goal is achieved by reformulating ***xor_abs***, ***latch_abs***, *cnt_abs*, and *cnt_appr* in $\mathcal{L}$ using $\Diamond$:

$$\begin{aligned}
xor_abs'(x,y,z) &\stackrel{df}{\equiv} \Diamond\forall t.\Diamond\iota\,(zt = xt + yt)\\
latch_abs'(d,q) &\stackrel{df}{\equiv} \Diamond\forall t_1,t_2.\Diamond\iota\,(next_abs\,(t_1\,,\,t_2) \supset q(t_2) = d(t_1))\\
cnt_abs'(x,y,z) &\stackrel{df}{\equiv} xor_abs'(x,z,y) \wedge latch_abs'(y,z)\\
cnt_appr'(x,y) &\stackrel{df}{\equiv} \Diamond\forall t_1,t_2.\Diamond\iota\,(next_abs\,(t_1\,,\,t_2) \supset y(t_2) = x(t_2) + y(t_1))
\end{aligned}$$

Syntactically speaking, all constraints are now removed from the formulae and replaced by $\Diamond$. Semantically speaking, and this is the crucial idea, a constraint now is no longer part of the proposition but of the proof. For instance,

$$\Diamond\forall t.\Diamond\iota\,(zt = xt + yt) \tag{4.22}$$

does not give any more information regarding constraints than indicating that there *may be* hidden assumptions, namely one for each instance of the $\Diamond$-operator. It is the proof of (4.22) that actually determines these constraints. In fact, the constraints depend on from which low-level axioms about the exclusive-or gate the abstracted proposition (4.22) is derived, and by which abstraction process. Here

xor (*cf.* 2.3.2) will be used but one might take a more detailed description of the gate, *e.g.* with variable delays, and then of course some other constraints would result. Also, there may be more than one way to verify an abstract behaviour of a composite device from properties about its components and each may result in a different constraint.

Synchronous Abstraction of XOR and Latch

In the sequel we will give derivations

$$\iota(xor(x,z,y)) \;\vdash_{x,y,z}\; xor\_abs'(x,z,y)$$
$$\iota(latch(y,clk,z)) \;\vdash_{y,z}\; latch\_abs'(y,z)$$

and thus establish *that* and *to what extent* the exclusive-or and latch are implementations of the abstract components. In these derivations, which will use the additional hypothesis *let's-not-bother*, actual constraints for the $\Diamond$ place-holders inside *xor_abs'* and *latch_abs'* will be determined.

We begin with the derivation of

$$let's\text{-}not\text{-}bother,\ \iota(xor(x,z,y)) \;\vdash_{x,y,z}\; xor\_abs'(x,z,y)$$

or, more explicitly, of

$$let's\text{-}not\text{-}bother,\ \iota(xor(x,z,y)) \;\vdash_{x,y,z}\; \Diamond\forall t.\Diamond\iota(yt = xt + zt) \quad (4.23)$$

shown in Figure 4–9. The means for introducing constraints, of course, is the hypothesis *let's-not-bother* which is used twice, namely for hiding the input constraint $stable\,x\,\delta_X \;\wedge\; stable\,z\,\delta_X$ and the sampling constraint *tick t*. Instead of using *let's-not-bother* we could have worked from the separate hypotheses

$$\forall x,z.\Diamond\iota(stable\,x\,\delta_X \;\wedge\; stable\,z\,\delta_X) \qquad \forall t.\Diamond\iota(tick\ t)$$

which is more explicit but results in the same constraints. As can be seen in the lower part of Figure 4–9, an instance of the ι-rule is referring to a derivation of

$$xor(x,z,y),\ stable\,x\,\delta_X \wedge stable\,z\,\delta_X,\ tick\ t \;\vdash_{x,y,z,t}\; yt = xt + zt \quad (4.24)$$

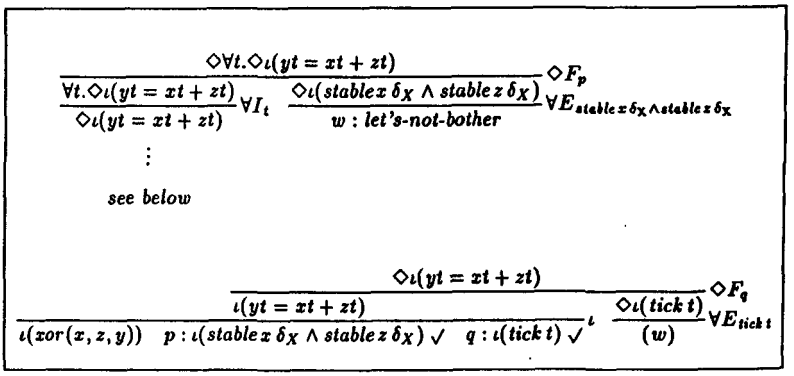


Figure 4–9: Derivation for Abstracting *xor* as a Synchronous Device

in $\mathcal{B}$. This is obtained by straightforward first-order reasoning, and really is where the main verification takes place. All the other proof steps in $\mathcal{L}$, *i.e.* all the rules shown in Figure 4-9 except ι, merely serve to associate the two invocations of *let's-not-bother*, indicated by variable w, with the two occurrences of $\diamond$. In other words, they serve to introduce the constraints into the right places. We remark that the derivation of (4.24) in $\mathcal{B}$ has to assume $\delta_X \geq 0$, and the following facts about integers and associated operations $\leq$, $-$:

$$\forall t, u.\ t - u + u = t \qquad \forall t.\ t \leq t \qquad \forall t, u : int.\ u \geq 0 \supset t - u \leq t$$

Let us check that the derivation of (4.23) in $\mathcal{L}$ indeed captures a derivation of

$$xor(x, z, y) \ \vdash_{x,y,z} \ (stable\, x\, \delta_X \wedge stable\, y\, \delta_X) \supset \forall t.\ tick\ t \supset yt = xt + zt$$

in $\mathcal{B}$ in the sense that the constraints are replaced by $\diamond$ and delegated to the constraint term. To this end we extract from the derivation given above in Figure 4-9 the constraint term X of type

$$|\diamond\forall t.\diamond\iota(yt = xt + zt)| \ \equiv\ \Omega^* \times (int \Rightarrow (\Omega^* \times int))$$

with a free variable w of type $|let's\text{-}not\text{-}bother|$ and v of type $|\iota(xor(x, z, y))| \equiv \mathbf{1}$.

We know that for this term the sequent

$$\textit{let's-not-bother}^{\#}\, w\,,\ (\iota\, xor(x,z,y))^{\#}\, v \ \vdash_{x,y,z,w,v}\ (\Diamond \forall t.\Diamond \iota(yt = xt + zt))^{\#}\, X$$

can be derived in $\mathcal{B}$, or, if we specialize w to ? to trivialize the hypothesis $\textit{let's-not-bother}^{\#}\, w$, and specialize v to $*$, the sequent

$$\iota(xor(x,z,y))^{\#}\, * \ \vdash_{x,y,z}\ (\Diamond \forall t.\Diamond \iota(yt = xt + zt))^{\#}\, (X\{?/w\}\{*/v\}).$$

Unrolling the definition of $^{\#}$ and using the equivalence $\phi^{c} \cong (\sqcap c) \supset \phi$ yields

$$xor(x,z,y) \ \vdash_{x,y,z}\ \sqcap(\pi_1 X\{?/w\}\{*/v\}) \supset \forall t.(\sqcap(\pi_1((\pi_2 X\{?/w\}\{*/v\})\, t)) \supset yt = xt + zt).$$

We leave it as an exercise to the reader to compute constraint term X and to verify

$$\sqcap(\pi_1 X\{?/w\}\{*/v\}) \ \cong\ \textit{stable}\, x\, \delta_X \wedge \textit{stable}\, y\, \delta_X \tag{4.25}$$

$$\sqcap(\pi_1((\pi_2 X\{?/w\}\{*/v\})\, t)) \ \cong\ \textit{tick}\ t. \tag{4.26}$$

This completes the synchronous abstraction for the *xor* gate. The other component to be abstracted is the *latch*, for which we can find a derivation of

$$\textit{let's-not-bother}\,,\ \iota(\textit{latch}\,(y, \textit{clk}, z)) \ \vdash_{y,z}\ \textit{latch\_abs}'(y,z)$$

or, more explicitly, of

$$\textit{let's-not-bother}\,,\ \iota(\textit{latch}\,(y, \textit{clk}, z)) \ \vdash_{y,z}\ \Diamond \forall t_1, t_2.\Diamond \iota(\textit{next\_abs}\,(t_1\,, t_2) \supset q(t_2) = q(t_1)) \tag{4.27}$$

swallowing the constraints in *latch_abs*. It is slightly more involved than the one for *xor* as, among other things, it requires that there be bounded induction on *int* in $\mathcal{B}$. We do not present the derivation here and state only that it contains the expected constraints. More precisely, the associated constraint term, call it L, of type

$$|\Diamond \forall t_1, t_2.\Diamond \iota(\textit{next\_abs}\,(t_1\,, t_2) \supset q(t_2) = q(t_1))|$$
$$\equiv\ \ \Omega^{*} \times (\textit{int} \Rightarrow \textit{int} \Rightarrow (\Omega^{*} \times 1))$$

with free variable w of type $|let's\text{-}not\text{-}bother|$ and v of type $\mathbf{1}$, is such that the $(\cdot)^\#$ translation of (4.27) obtains

$$latch(y, clk, z) \vdash_{y,z} (\sqcap(\pi_1 L\{?/w\}\{*/v\})) \supset \forall t_1, t_2.$$
$$(\sqcap((\pi_2 L\{?/w\}\{*/v\})\, t_1\, t_2)) \supset next\_abs(t_1, t_2) \supset q(t_2) = q(t_1)$$

in $\mathcal{B}$, and further it holds that

$$\sqcap(\pi_1 L\{?/w\}\{*/v\}) \cong one\_shot \wedge min\_sep\, \delta_L \tag{4.28}$$
$$\sqcap((\pi_2 L\{?/w\}\{*/v\})\, t_1\, t_2) \cong tick\, t_1 \wedge tick\, t_2. \tag{4.29}$$

Abstract Verification of the Modulo-2 Counter

As the constraint-free version of (4.21) we now set out to derive the sequent

$$cnt\_abs'(x, y, z) \vdash_{x,y,z} cnt\_appr'(x, y) \tag{4.30}$$

which differs from the ideal verification goal (2.4) only in the presence of $\Diamond$ and ι. Now let us introduce the following syntactic abbreviations:

$$\theta \stackrel{df}{\equiv} next\_abs(t_1, t_2)$$
$$\epsilon \stackrel{df}{\equiv} y(t_2) = x(t_2) + y(t_1)$$
$$\phi \stackrel{df}{\equiv} yt = xt + zt$$
$$\psi \stackrel{df}{\equiv} z(t_2) = y(t_1)$$

With these abbreviations (4.30) translates into the sequent

$$\Diamond\forall t.\Diamond\iota\,\phi \wedge \Diamond\forall t_1, t_2.\Diamond\iota(\theta \supset \psi) \vdash_{x,y,z} \Diamond\forall t_1, t_2.\Diamond\iota(\theta \supset \epsilon) \tag{4.31}$$

Figure 4-10 shows the complete natural deduction tree for (4.31), where rule ι depends on a derivation of $\phi\{t_2/t\}$, $\psi \vdash_{x,y,z,t_1,t_2} \epsilon$ in $\mathcal{B}$, *i.e.* of

$$y(t_2) = x(t_2) + z(t_2),\ z(t_2) = y(t_1) \vdash_{x,y,z,t_1,t_2} y(t_2) = x(t_2) + y(t_1)$$

which is simple enough. The derivation of (4.31) proves that composing a delay-

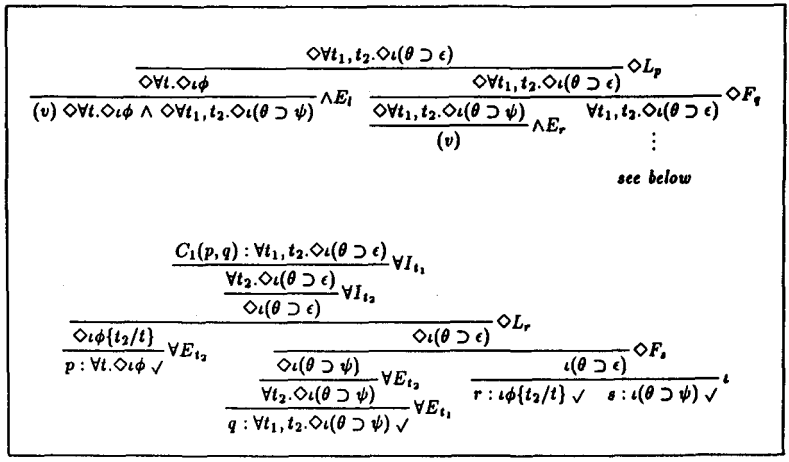


Figure 4–10: Verification of Correctness for *Modulo*-2 Counter

free *modulo*-2 sum and an one-unit delay as in Figure 2–3 yields a stoppable *modulo*-2 counter. The reader may convince themselves that the only steps in this derivation that would not arise in an ideal proof, *i.e.* one which does not consider constraints at all, are the occurrences of the $\diamond L, \diamond F$ rules. Further, one notices, that these extra rules are used in a canonical way, *viz.* to implement the derived inference rule

$$\frac{\Gamma, \diamond M_1, \ldots, \diamond M_k \vdash_\Delta \diamond M}{\Gamma, M_1, \ldots, M_k \vdash_\Delta M}\diamond F^*$$

which generalizes rule $\diamond F$. It has the effect that at any point in a derivation the assertion and an arbitrary number of hypotheses can be prefixed by $\diamond$. This is done twice in Figure 4–10, once in the upper and once in the lower half of the tree.

The derivation of (4.31) can be called abstract since it is performed without looking at or even manipulating explicit constraints. The $\diamond$s only indicate where constraints are to be expected and so intuitively serve as place-holders for constraints. In manipulating the place-holder instead of real constraints the

derivation is independent of constraints, and yet the associated proof term

$$C \stackrel{df}{\equiv} \Diamond L_p\,(\wedge E_l\,v\,,\ \Diamond F_q\,(\wedge E_r\,v\,,\ \forall I_{t_1}\,\forall I_{t_2}\,\Diamond L_r\,(\forall E_{t_2}\,p\,,\ \Diamond F_s\,(\forall E_{t_2}\,\forall E_{t_1}\,q\,,\ \iota\,(r,s)))))$$

where v refers to the hypothesis of (4.31), retains enough information for extracting the constraints inside $cnt\_appr'\,(x,y)$ out of those in $cnt\_abs'\,(x,y,z)$. This now can be done in a completely separate phase: in the *constraint analysis* phase.

First, we translate C into a constraint term applying the translation of Figure 3–16:

$$\begin{aligned}
C &\equiv \Diamond L_p\,(\pi_1 v\,,\ \Diamond F_q\,(\pi_2 v\,,\ \lambda t_1.\,\lambda t_2.\,\Diamond L_r\,(p\,t_2\,,\ \Diamond F_s\,(q\,t_1\,t_2\,,\ *))))\\
&\equiv \Diamond L_p\,(\pi_1 v\,,\ \Diamond F_q\,(\pi_2 v\,,\ \lambda t_1.\,\lambda t_2.\,\Diamond L_r\,(p\,t_2\,,\ (\pi_1(q\,t_1\,t_2)\,,\ *))))\\
&\equiv \Diamond L_p\,(\pi_1 v\,,\ \Diamond F_q\,(\pi_2 v\,,\ \lambda t_1.\,\lambda t_2.\,(\pi_1(\pi_1(q\,t_1\,t_2)\,,\ *)\,@\,\pi_1(p\,t_2)\,,\ \pi_2(p\,t_2))))\\
&\equiv \Diamond L_p\,(\pi_1 v\,,\ (\pi_1(\pi_2 v)\,,\ \lambda t_1.\,\lambda t_2.\,(\pi_1(\pi_1((\pi_2(\pi_2 v))\,t_1\,t_2)\,,\ *)\,@\,\pi_1(p\,t_2)\,,\\
&\qquad \pi_2(p\,t_2))))\\
&\equiv (\pi_1(\pi_1(\pi_2 v)\,,\ \lambda t_1.\,\lambda t_2.\,(\pi_1(\pi_1((\pi_2(\pi_2 v))\,t_1\,t_2)\,,\ *)\,@\,\pi_1((\pi_2(\pi_1 v))\,t_2)\,,\\
&\qquad \pi_2((\pi_2(\pi_1 v))\,t_2)))\ @\ \pi_1(\pi_1 v)\ ,\ \pi_2(\pi_1(\pi_2 v)\,,\\
&\qquad\quad \lambda t_1.\,\lambda t_2.\,(\pi_1(\pi_1((\pi_2(\pi_2 v))\,t_1\,t_2)\,,\ *)\,@\,\pi_1((\pi_2(\pi_1 v))\,t_2)\,,\\
&\qquad\qquad \pi_2((\pi_2(\pi_1 v))\,t_2))))
\end{aligned}$$

This term partly can be 'evaluated' via β-equalities to obtain

$$\begin{aligned}
C \;=\; &(\pi_1(\pi_2 v)\,@\,\pi_1(\pi_1 v)\ ,\ \lambda t_1.\,\lambda t_2.\,(\pi_1((\pi_2(\pi_2 v))\,t_1\,t_2)\,@\,\pi_1((\pi_2(\pi_1 v))\,t_2)\,,\\
&\pi_2((\pi_2\pi_1 v)\,t_2))).
\end{aligned}$$

By the correctness theorems of constraint extraction we know that C is a term of type $\Omega^* \times (int \Rightarrow int \Rightarrow (\Omega^* \times 1))$ if v free variable of type $(\Omega^* \times (int \Rightarrow (\Omega^* \times 1))) \times (\Omega^* \times (int \Rightarrow int \Rightarrow (\Omega^* \times 1)))$ such that the sequent

$$(cnt\_abs'(x,y,z))^{\#}\,v \ \vdash_{x,y,z,v}\ (cnt\_appr'(x,y))^{\#}\,C$$

is derivable in $\mathcal{B}$. With a little massaging, this sequent can be shown to be equivalent to

$$\begin{array}{l} \delta_1 \supset \forall t.\,(\gamma_1\, t) \supset \phi \ \wedge \\ \quad \delta_2 \supset \forall t_1, t_2.\,(\gamma_2\, t_1\, t_2) \supset \theta \supset \psi \ \vdash_\Delta \\ \quad\quad (\delta_1 \wedge \delta_2) \supset \forall t_1, t_2.\,(\gamma_1\, t_2 \wedge \gamma_2\, t_1\, t_2) \supset \theta \supset \epsilon \end{array} \qquad (4.32)$$

where $\Delta = x, y, z, \delta_1, \delta_2, \gamma_1, \gamma_2$ with δ_1, δ_2 variables of type Ω, γ_1 variable of type $int \Rightarrow \Omega$, and γ_2 variable of type $int \Rightarrow int \Rightarrow \Omega$. The proof of this, which is entirely straightforward, is omitted. It uses the definition of $(\cdot)^\sharp$, the above evaluation of C, and the equivalence $\phi^{[\gamma_k,\ldots,\gamma_1]} \cong \sqcap[\gamma_k, \ldots, \gamma_1] \supset \phi \cong \gamma_1 \wedge \cdots \wedge \gamma_k \supset \phi$ for $k = 1$ and $k = 2$.

So, what we have got from working in the extracted constraint term C into the abstract sequent (4.31) is a modification (4.32) that tells us how any set of constraints for subcomponents can be translated into constraints for the composite circuit. More precisely, if δ_1 and γ_1 are constraints on the input and sampling times for *latch*, respectively, and δ_2, γ_2 such constraints for *latch*, then $\delta_1 \wedge \delta_2$ and $\gamma_1\, t_2 \wedge \gamma_2\, t_1\, t_2$ are sufficient constraints on the inputs and sampling times, respectively, for the *modulo*-2 counter. Note, the computation of constraints is captured completely within C, the translation of (4.31) into (4.32) only establishes that the translation, and thus C, has the required properties.

* * *

It is important to realize that if we had had to confine reasoning to the base logic, then the sequent (4.32) is the best we could have achieved in order to be constraint-independent. However, deriving (4.32) directly would require us to mess around with the 'abstract' constraints $\delta_1, \delta_2, \gamma_1, \gamma_2$ and to throw in extra proof steps to deal with them. In contrast, using lax logic and constraint extraction the sequent (4.32) was obtained, up to equivalence, in a completely automatic fashion from a natural and constraint-free proof.

Concrete-Level Composition of the Counter

The abstract verification that composition of delay-free *modulo*-2 sum and one-unit delay satisfies the specification of a *modulo*-2 counter has been achieved through the derivation of

$$cnt\_abs'(x,y,z) \;\vdash_{x,y,z}\; cnt\_appr'(x,y). \tag{4.33}$$

Further, the derivations of

$$let's\text{-}not\text{-}bother,\ \iota(xor(x,z,y)) \;\vdash_{x,y,z}\; xor\_abs'(x,z,y) \tag{4.34}$$

$$let's\text{-}not\text{-}bother,\ \iota(latch(y,clk,z)) \;\vdash_{y,z}\; latch\_abs'(y,z) \tag{4.35}$$

witness that in a synchronous environment *xor* and *latch* may be regarded as implementations of the corresponding abstract components delay-free *modulo*-2 sum and one-unit delay. From these three parts (4.33), (4.34), and (4.35) we extracted the constraint terms C, X, L, respectively. Residing inside X and L, through the reference to hypothesis *let's-not-bother*, are certain behavioural constraints which record assumptions about the environment of the components under which their abstraction is possible. The constraint term C contains the information of how these constraints are to be composed to obtain the constraints for the *modulo*-2 counter, which is captured by the sequent (4.32) formally.

Now we may put pieces together and prove that in a synchronous environment the composition of concrete-level components *xor* and *latch* can be regarded as an implementation of the abstract *modulo*-2 counter. This comes down to a derivation of

$$let's\text{-}not\text{-}bother,\ \iota(xor(x,z,y)) \wedge \iota(latch(y,clk,z)) \;\vdash_{x,y,z}\; cnt\_appr'(x,y) \tag{4.36}$$

which is obtained easily by composing (4.33), (4.34), and (4.35) in $\mathcal{L}$ as shown in Figure 4-11 The hypothesis *let's-not-bother* on which derivations (4.34) and (4.35) depend is not shown in Figure 4-11. Let us now examine the constraints

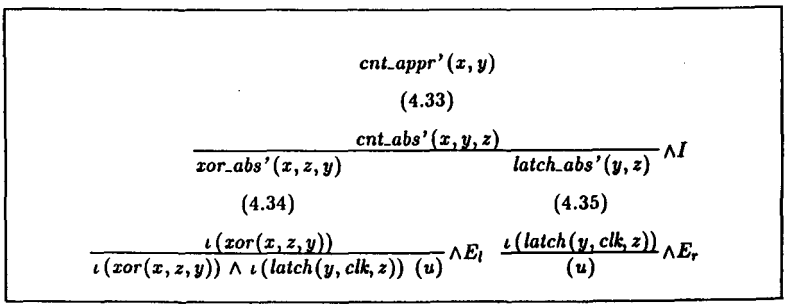


Figure 4–11: Low-Level Implementation of *Modulo*-2 Counter

residing in the derivation of Figure 4–11. As the associated constraint term, called I, we obtain:

$$\begin{aligned} I &\equiv C\{\wedge I\,(X\{\wedge E_l\,u/v\}\,,\,L\{\wedge E_r\,u/v\})/v\} \\ &\equiv C\{(X\{\pi_1 u/v\}\,,\,L\{\pi_2 u/v\})/v\} \end{aligned}$$

I has a free variable u of type $|\iota\,(xor\ (x,z,y))\ \wedge\ \iota\,(latch\ (y,clk,z))| \equiv \mathbf{1} \times \mathbf{1}$, and variable w of type $|let's\text{-}not\text{-}bother|$ which is free in sub-terms X and L. By correctness of constraint extraction we conclude that the sequent

$$\begin{aligned} &(let's\text{-}not\text{-}bother)^{\#}\,w\,,\,(\iota\,(xor(x,z,y))\ \wedge\ \iota\,(latch\,(y,clk,z)))^{\#}\,u \\ &\quad \vdash_{x,y,z,w,u}\ (cnt\_appr'(x,y))^{\#}\,I \end{aligned}$$

is derivable in $\mathcal{B}$. If we specialize w to ? and u to $(*,*)$, then it can be shown that this sequent simplifies to become

$$\begin{aligned} &xor(x,z,y)\ \wedge\ latch\,(y,clk,z) \\ &\quad \vdash_{x,y,z}\ \sqcap((\pi_1 X\{?/w\}\{*/v\})\,@\,(\pi_1 L\{?/w\}\{*/v\}))\ \supset \\ &\qquad \forall t_1,t_2.\ \sqcap\,(\pi_1((\pi_2 X\{?/w\}\{*/v\})\,t_2)\,@\,\pi_1((\pi_2 L\{?/w\}\{\pi_2 u/v\})\,t_1\,t_2))\ \supset \\ &\qquad\quad \theta\ \supset\ \epsilon. \end{aligned}$$

Thus, using the equivalences (4.25), (4.26), (4.28), (4.29), the input constraint for the *modulo*-2 counter is

$$\sqcap((\pi_1 X\{?/w\}\{*/v\})\,@\,(\pi_1 L\{?/w\}\{*/v\}))$$

$$\cong \quad one\_shot \wedge min\_sep\,\delta_L \wedge stable\,x\,\delta_X \wedge stable\,y\,\delta_X$$

and the sampling constraint is

$$\sqcap(\pi_1((\pi_2 X\{?/w\}\{*/v\})\,t_2) \mathbin{@} \pi_1((\pi_2 L\{?/w\}\{\pi_2 u/v\})\,t_1\,t_2))$$
$$\cong \quad tick\,t_1 \wedge tick\,t_2 \wedge tick\,t_2.$$

This shows that the constraint term I has in fact collected together the constraints for *xor* and *latch*. The reader is invited to compare this with sequent (4.32).

* * *

Chapter 5

Some Meta-Theory for Lax Logic

In Section 5.1 we investigate what properties the notion of constraint is to have if it is to serve as a Kripke-style semantics for the modal operator $\Diamond$, or in other words: to what extent it is justifiable to call $\Diamond$ a 'modal' operator in the standard Kripke sense. In Section 5.2 a category theoretical interpretation of lax logic is presented that provides a more semantical explanation of the constraint extraction process and of its correctness.

5.1 On Kripke Semantics for $\Diamond$

Our aim in this section is to analyze, in terms of ordinary Kripke-style semantics, the intuitive reading according to which $\Diamond M$ means 'under some constraint', M. We naively assume the constraints form a set $\mathbf{C}$, equipped with a binary accessibility relation R that is used as a Kripke *frame* on which 'truth' of formulae is decided.

As a simple start let us focus on the following propositional modal logic: *Formulae* M are generated by the grammar

$$M ::= \phi \mid \Diamond M \mid M \wedge M$$

where ϕ stands for an atomic sentence. We assume that there are at least two distinct atomic sentences. A *sequent* is of the form

$$M_1, \ldots, M_k \vdash M$$

where M_i, $i = 1, \ldots, k$, and M are formulae. The *inference rules* are $\Diamond I$, $\Diamond F$, $\Diamond M$, $\wedge E_l$, $\wedge E_r$, $\wedge I$, and the structural rules *id*, *weak*, *perm*, and *cut* of Figure 3–7.

According to classical Kripke semantics a truth *valuation* on a frame $\mathbf{C}$ is given by a function V that associates with every atomic sentence ϕ a subset $V(\phi) \subseteq \mathbf{C}$, *viz.* the set of constraints under which ϕ is deemed to hold. Such a triple $\mathcal{C} = (\mathbf{C}, R, V)$ is called a Kripke *model*. A formula M then is *valid at* a constraint c *in* model $\mathcal{C}$, written $\mathcal{C}, c \models M$, if M is of shape

- ϕ and $c \in V(\phi)$
- $\Diamond N$ and there exists an a, cRa, such that $\mathcal{C}, a \models N$
- $N \wedge K$ and both $\mathcal{C}, a \models N$ and $\mathcal{C}, a \models K$.

A formula M is *valid in* model $\mathcal{C}$, written $\mathcal{C} \models M$, if for all constraints $c \in \mathbf{C}$, one has $\mathbf{C}, c \models M$.

The notion of validity is lifted to sequents and rules in the obvious way: A sequent $M_1, \ldots, M_k \vdash M$ is *valid in* model $\mathcal{C}$ if for all constraints c, whenever all hypotheses M_i, $i = 1, \ldots, k$, are valid at c in $\mathcal{C}$, then the assertion M is valid at c in $\mathcal{C}$.

Finally, a rule is said to be *valid* in a class of models if whenever all premisses of the rule are valid in each model of the class, then the conclusion of the rule is valid in each model of the class.

Given these definitions we may now ask what conditions on a frame must be imposed in order for the rules $\Diamond I$, $\Diamond M$ and $\Diamond F$ to be valid in the class of models based on this frame.

Lemma 5.1.1 *Let* $(\mathbf{C}, R)$ *be a fixed frame and let* $\mathcal{M}$ *be the class of models* $(\mathbf{C}, R, V)$ *where* V *is an arbitrary valuation on* R.

- *Rule* $\Diamond I$ *is valid in* $\mathcal{M}$ *iff* R *is reflexive*
- *Rule* $\Diamond M$ *is valid in* $\mathcal{M}$ *iff* R *is transitive*
- *Rule* $\Diamond F$ *is valid in* $\mathcal{M}$ *iff* R *is discrete, i.e. for all* $c, d \in \mathbf{C}$, *if* cRd, *then* $c = d$.

Proof: The first two statements concerning $\Diamond I$ and $\Diamond M$ are well-known, *cf.* [Che80] for instance. The interesting case is rule $\Diamond F$; let $\Gamma = \phi$, $M = \psi$, and $N = \phi \wedge \psi$, where ϕ, ψ are arbitrary, but distinct atomic sentences. Thus, the instance of $\Diamond F$ that we are looking at is

$$\frac{\phi, \Diamond\psi \vdash \Diamond(\phi \wedge \psi)}{\phi, \psi \vdash \phi \wedge \psi} \Diamond F$$

It is easy to check that the premiss is valid in all models, so for $\Diamond F$ to be valid in $\mathcal{M}$ we must have that the sequent

$$\phi, \Diamond\psi \vdash \Diamond(\phi \wedge \psi)$$

is valid in all models $(\mathbf{C}, R, V)$ in $\mathcal{M}$. The valuation V in the sense defined may send both sentences ϕ, ψ to arbitrary subsets $A := V(\phi)$ and $B := V(\psi)$ of $\mathbf{C}$. Unravelling the definition of a sequent being valid shows that in order for this particular instance to be valid for all valuations the following second-order condition must be met:

$$\begin{aligned} &\forall A, B \subseteq \mathbf{C}.\, \forall c \in \mathbf{C}. \\ &\quad c \in A \;\&\; (\exists y \in \mathbf{C}.\, cRy \;\&\; y \in B) \\ &\quad\Rightarrow\; \exists y \in \mathbf{C}.\, cRy \,\&\, y \in A \,\&\, y \in B \end{aligned}$$

This condition now implies that R is discrete. Namely, if $a, b \in \mathbf{C}$ are given, specialize $A = \{a\}$ and $B = \{b\}$ for which the condition becomes

$$\begin{aligned} &\forall c \in \mathbf{C}.\, c \in \{a\} \;\&\; (\exists y \in \mathbf{C}.\, cRy \,\&\, y \in \{b\}) \\ &\quad\Rightarrow\; \exists y \in \mathbf{C}.\, cRy \,\&\, y \in \{a\} \,\&\, y \in \{b\} \end{aligned} \tag{5.1}$$

which is equivalent to $aRb \Rightarrow a = b$. Thus, discreteness of R is necessary for the validity of $\Diamond F$. That it is also a sufficient condition is seen as follows: If R is discrete, then, for every valuation, a formula M is valid at c *iff* $\Diamond M$ is valid at c. Thus, the meaning of $\Diamond$ collapses and both sides of the rule bar in $\Diamond F$ have the same semantical meaning. ∎

Since we want to stick to rule $\Diamond F$, Lemma 5.1.1 means that the classical notion of valuation is not feasible for our purposes. Validity of $\Diamond F$ forces the accessibility relation of a constraint frame $(\mathbf{C}, R)$ to degenerate. This is unfortunate as R then cannot be used to model a non-trivial relationship between constraints, and the meaning of $\Diamond$ as a modal operator becomes vacuous.

From the proof of the lemma it is clear that the reason of this lies in the fact that the classical notion of valuation allows truth values $V(\phi)$ of atomic sentences to be arbitrary subsets of $\mathbf{C}$ that do not need to bear any relation to R. In contrast, an *intuitionistic* notion of truth would require the sets $V(\phi)$ to be hereditary, *i.e.* if $c \in V(\phi)$ and cRd, then $d \in V(\phi)$. Arbitrary elements $a, b \in \mathbf{C}$ can no longer be fully determined by truth values of propositions. Thus, in the proof of Lemma 5.1.1 it would not be possible to choose for A, B the singleton sets $\{a\}$, $\{b\}$ but only sets

$$A = \{x \mid aRx\} \qquad B = \{x \mid bRx\}$$

say, so that in place of (5.1) we get

$$\begin{aligned} &\forall c \in \mathbf{C}.\ aRc\ \&\ (\exists y \in \mathbf{C}.\ cRy\ \&\ bRy) \\ &\quad \Rightarrow\ \exists y \in \mathbf{C}.\ cRy\ \&\ aRy\ \&\ bRy \end{aligned}$$

which is trivially satisfied whenever R is transitive.

In fact, the intuitionistic notion of valuation appears quite natural in our setting if accessibility is taken to express the *strength* of constraints: cRd if d is *stronger* than c. Then, if the strength of c is reflected by the class $\Omega_c = \{\phi \mid$

$c \in V(\phi)\}$ of atomic sentences that are true 'under constraint' c, one naturally arrives at the condition $cRd \Rightarrow \Omega_c \subseteq \Omega_d$, or, $cRd \,\&\, c \in V(\phi) \Rightarrow d \in V(\phi)$.

From this discussion it appears that a Kripke style semantic for our modal operator $\Diamond$ may be possible for an intuitionistic notion of valuation based on constraint frames with reflexive and transitive accessibility relation. Indeed, it will be seen that at least for the closed propositional fragment of $\mathcal{L}$ all our rules can be justified, under an intuitionistic interpretation, on non-degenerate frames.

First we need to make a few definitions. The fragment of $\mathcal{L}$ that we are going to investigate, denoted by $\mathcal{L}_0$, has as *well-formed formulae*

$$M ::= \iota\phi \mid \Diamond M \mid M \wedge M \mid M \supset M \mid M \vee M \mid \mathit{true} \mid \mathit{false}$$

where ϕ stands for a well-formed closed proposition of $\mathcal{B}$. The rules of $\mathcal{L}_0$ are all those of $\mathcal{L}$, as in Figures 3-7 3-9, that pertain to the closed propositional fragment. In particular, we have rule ι in $\mathcal{L}_0$, so that $\mathcal{L}_0$ contains $\mathcal{B}$ as a sub-logic.

Definition 5.1.2

- *A* constraint frame *is a preordered set* $(\mathbf{C}, \sqsubseteq)$.
- *A* constraint valuation for $\mathcal{B}$ *on a constraint frame* $(\mathbf{C}, \sqsubseteq)$ *is a map* V *that associates with every well-formed closed proposition* ϕ *of* $\mathcal{B}$ *an upper closed subset* $V(\phi)$ *of* $\mathbf{C}$, *i.e. for all* $c, d \in \mathbf{C}$, *such that* $c \sqsubseteq d$, *if* $c \in V(\phi)$ *then* $d \in V(\phi)$. *Also,* V *is to respect entailment of* $\mathcal{B}$, *i.e. if for closed* $\phi_1, \ldots, \phi_n, \psi$ *we have* $\phi_1, \ldots, \phi_n \vdash \psi$, *then* $\bigcap_i V(\phi_i) \subseteq V(\psi)$.
- *A* constraint model *for* $\mathcal{B}$ *then is a triple* $(\mathbf{C}, \sqsubseteq, V)$ *where* $(\mathbf{C}, \sqsubseteq)$ *is a constraint frame and* V *a constraint valuation for* $\mathcal{B}$ *on* $(\mathbf{C}, \sqsubseteq)$.

The elements of a constraint frame $\mathbf{C}$ are called *constraints*, ranged over by a, b, c, *etc.* , with $a \sqsubseteq b$ meaning that b is 'stronger' than a. In the sequel we will simply

talk about constraint models rather than constraint models *for* $\mathcal{B}$. Constraint models will be used to interpret formulae of $\mathcal{L}_0$ by combining the ways Kripke frames interpret modal and intuitionistic logic.

Definition 5.1.3 *Let* $\mathcal{C} = (\mathbf{C}, \sqsubseteq, V)$ *be a constraint model for* $\mathcal{B}$. *Given a formula* M *of* $\mathcal{L}_0$ *and a constraint* $c \in \mathbf{C}$, *we say* M *is* valid at c *in* $\mathcal{C}$, *written* $\mathcal{C}, c \models M$ *iff*

- M *is of form* $\Diamond N$ *and there exists a constraint* a *in* $\mathbf{C}$ *with* $c \sqsubseteq a$ *such that* $\mathcal{C}, a \models N$
- M *is* $\iota\phi$ *and* $c \in V(\phi)$
- M *is* $N \wedge K$ *and both* $\mathcal{C}, c \models N$ *and* $\mathcal{C}, c \models K$
- M *is* $N \vee K$ *and* $\mathcal{C}, c \models N$ *or* $\mathcal{C}, c \models K$
- M *is true*
- M *is* $N \supset K$ *and for all* a *such that* $c \sqsubseteq a$, $\mathcal{C}, a \models N$ *implies* $\mathcal{C}, a \models K$

A formula M *is said to be* valid in $\mathcal{C}$, *written* $\mathcal{C} \models M$ *if for all* $c \in \mathbf{C}$, M *is valid at* c *in* $\mathcal{C}$; M *is* valid *if* M *is valid in any constraint model* $\mathcal{C}$.

Notice, the definition covers the case $M \equiv$ *false*: for no $\mathcal{C}$ and c we have $\mathcal{C}, c \models$ *false*. Wherever the constraint model $\mathcal{C}$ is understood we will simply write $c \models M$ rather than $\mathcal{C}, c \models M$. A first result that we want to have for our notion of validity is that the *modal-free* formulae, i.e. those which do not contain $\Diamond$, behave like in ordinary intuitionistic logic.

Lemma 5.1.4 *Let* $\mathcal{C}$ *be a constraint model and* M *a modal-free formula. If* M *is valid in* $\mathcal{C}$ *at some constraint* c *and* $c \sqsubseteq b$, *then* M *is valid at* b, *too.*

Proof: Suppose $c \models M$, and $c \sqsubseteq b$. We prove $b \models M$ by induction on the structure of M. If $M \equiv \iota\phi$ then the assumption gives $c \in V(\phi)$ which

implies $b \in V(\phi)$ by the hereditary property of constraint valuations. Suppose $M \equiv N \supset K$. We have to show $a \models N$ implies $a \models K$ for all a, $b \sqsubseteq a$. But this follows from the assumption $c \models N \supset K$ since by transitivity, $c \sqsubseteq a$. The remaining cases for M are trivial. ∎

As before we lift the notion of validity to sequences and rules in the following way: A sequent $M_1, \ldots, M_k \vdash M$ is *valid* in a constraint model $\mathcal{C}$ if for every c, whenever all hypotheses M_i, $i = 1, \ldots, k$, are valid at c in $\mathcal{C}$ then the assertion M is valid at c in $\mathcal{C}$. A rule is *valid* (in a class of constraint models) if whenever all premisses of the rule are valid in every constraint model (of the class) then the conclusion of the rule is valid in every model (of the class).

Lemma 5.1.5 *All rules of $\mathcal{L}_0$ are valid, where in rules $\Diamond F$ and $\supset I$ the side condition is imposed that all 'passive' hypotheses Γ are modal-free.*

Proof: Let a constraint model $\mathcal{C}$ be given. We begin with rule $\Diamond I$. We have to show that for all well-formed M the sequent $M \vdash \Diamond M$ is valid in $\mathcal{C}$. To this end assume a c with $\mathcal{C}, c \models M$. We have to show $\mathcal{C}, c \models \Diamond M$, *i.e.* that there exists a constraint a, $c \sqsubseteq a$ such that $\mathcal{C}, a \models M$. Simply choose $a = c$ and use reflexivity of $\sqsubseteq$.

To verify that rule $\Diamond M$ is valid in $\mathcal{C}$ we assume a c such that $\mathcal{C}, c \models \Diamond\Diamond M$. By definition this means there are constraints a, b, with $c \sqsubseteq b$, $b \sqsubseteq a$ and $\mathcal{C}, a \models M$. By transitivity $c \sqsubseteq a$, so $\mathcal{C}, c \models \Diamond M$.

Now consider the rule $\Diamond F$. For it to be valid in $\mathcal{C}$ any instance of a sequent $M_1, \ldots, M_k, \Diamond M \vdash \Diamond N$ has to be valid in $\mathcal{C}$ under the supposition that $M_1, \ldots, M_k, M \vdash N$ is valid in $\mathcal{C}$. So, let a constraint c be given and assume $\mathcal{C}, c \models M_i$, $i = 1, \ldots, k$, and $\mathcal{C}, c \models \Diamond M$. The latter condition means there exists a, $c \sqsubseteq a$, such that $\mathcal{C}, a \models M$. We are done if we can show that also $(*)$ $\mathcal{C}, a \models M_i$, $i = 1, \ldots, k$. For then the supposition that $M_1, \ldots, M_k, M \vdash N$ is valid can be used to conclude that $\mathcal{C}, a \models N$, whence $\mathcal{C}, c \models \Diamond N$. Now, because of the side condition imposed we may assume that all the additional 'passive' hypotheses M_i are modal-free. Thus, $(*)$ follows from Lemma 5.1.4.

Rule ι is seen to be valid as follows: Suppose in $\mathcal{B}$ we have $\phi_1, \ldots, \phi_n \vdash \psi$. We have to show that $\iota\phi_1, \ldots, \iota\phi_n \vdash \iota\psi$ is valid in $\mathcal{C}$. If c is a constraint, and $\mathcal{C}, c \models \iota\phi_i$ for all $i = 1, \ldots, n$, then by definition $c \in \bigcap_i V(\phi_i)$. By the properties of valuations then, $c \in V(\psi)$, whence $\mathcal{C}, c \models \iota\psi$.

The remaining structural rules, and with the exception of $\supset I$ the rules for the ordinary propositional connectives are trivial to check. As to $\supset I$, it has to be shown that under the supposition, that $M_1, \ldots, M_k, M \vdash N$ is valid, the sequent $M_1, \ldots, M_k \vdash M \supset N$ is valid, too. So, let c be a constraint and assume $\mathcal{C}, c \models M_i$, $i = 1, \ldots, k$. Given the definition of validity for $\supset$, it has to be demonstrated that for all a, $c \sqsubseteq a$, if $\mathcal{C}, a \models M$, then $\mathcal{C}, a \models N$. This follows from the supposition provided we we can show for all $i = 1, \ldots, k$, $\mathcal{C}, a \models M_i$. This is the same situation as in proving $\Diamond F$: We know the M_i are valid at c and need to verify they are valid at the stronger constraint a. As before we make use of the side-condition that all M_i are modal-free and invoke Lemma 5.1.4. ∎

It is possible to show that for arbitrary constraint models the side-condition in rules $\Diamond F$ and $\supset I$ to the effect that all 'passive' hypotheses Γ be non-modal formulae cannot be dropped. In the following the question will be investigated of what property of the constraint frame is needed to get rid of the side condition. First observe that the sequent

$$\Diamond F^0 \qquad K \supset (M \supset N) \vdash (K \supset (\Diamond M \supset \Diamond N)),$$

which may be considered a 'flattened' version of rule $\Diamond F$ where the rule bar has been reduced to the turnstile $\vdash$ and the turnstile replaced by implication $\supset$, is valid in every constraint model. To see this let d be a constraint and suppose we have $(*)$ $d \models K \supset (M \supset N)$. We wish to show $d \models (K \supset (\Diamond M \supset \Diamond N))$, so let c, $d \sqsubseteq c$, such that $c \models K$, and further b, $c \sqsubseteq b$, with $b \models \Diamond M$, be given. It suffices to show that also $b \models \Diamond N$. First note that since $d \sqsubseteq c$ and $c \models K$, $(*)$ obtains $(**)$ $c \models M \supset N$. Since $b \models \Diamond M$, there must be a, $b \sqsubseteq a$, such that $a \models M$. Now by transitivity, $c \sqsubseteq a$, so we can conclude from $(**)$, $a \models N$. But this means nothing but $b \models \Diamond N$.

Note, in $\Diamond F^0$ the formulae K may be any formula, *i.e.* it need not be modal-free while in $\Diamond F$ there is the side condition that all hypotheses in context Γ be modal-free. This difference, of course, is not contradictory as both versions are interderivable only by using the rule $\supset I$, for which the same restriction applies as for $\Diamond F$. So, let us analyze the validity of rule $\supset I$ to see what we can do. Consider the following special instance of $\supset I$:

$$\frac{N \vdash \mathit{true} \supset N}{N, \mathit{true} \vdash N}$$

where N is any well-formed formula (of $\mathcal{L}_0$). Certainly, the premiss $N, \mathit{true} \vdash N$ is valid in every constraint model, hence for this rule to be valid the sequent $N \vdash \mathit{true} \supset N$ must be valid in every constraint model $\mathcal{C} = (\mathbf{C}, \sqsubseteq, V)$. Using the definitions this is seen to be equivalent to

$$\forall c, d \in \mathbf{C}.\ c \sqsubseteq d \,\&\, c \models N \ \Rightarrow\ d \models N$$

which is precisely saying that the set $V(N) = \{c \mid c \models N\}$ is hereditary, *i.e.* upper closed. Thus, we have found a necessary condition for rule $\supset I$ to be valid without side-condition; it is also sufficient as we know from the proof of Lemma 5.1.5.

Lemma 5.1.6 *Let $\mathcal{M}$ be a class of constraint models. Rule $\supset I$ without side condition is valid in $\mathcal{M}$ iff for all models $\mathcal{C} = (\mathbf{C}, \sqsubseteq, V)$ in $\mathcal{M}$ and all formulae N, the set $V(N) = \{c \mid c \models N\}$ is upper closed wrt. $\sqsubseteq$.*

The conclusion from this is that if we want to drop the side-condition on rules $\Diamond F$ and $\supset I$ we must restrict the class of constraint models to those for which Lemma 5.1.4 holds for *all* formulae, not just for the modal-free. If this class again is to be characterized by a condition on the constraint frame, we end up with

Definition 5.1.7 *A constraint frame $(\mathbf{C}, \sqsubseteq)$ is called* confluent *iff for all $d, c, b \in \mathbf{C}$ with $d \sqsubseteq c$ and $d \sqsubseteq b$ there exists a, such that $c \sqsubseteq a$ and $b \sqsubseteq a$. A constraint model $\mathcal{C} = (\mathbf{C}, \sqsubseteq, V)$ is* confluent *if its frame is.*

Lemma 5.1.8 *Let* $(\mathbf{C}, \sqsubseteq)$ *be a constraint frame. Then, for all formulae* M *and all models* $(\mathbf{C}, \sqsubseteq, V)$ *based on this frame, i.e. for which* V *is any valuation in the sense of Definition 5.1.2, the set*

$$V(M) \stackrel{df}{=} \{c \mid c \models M\}$$

is upper closed wrt. $\sqsubseteq$ *iff* $(\mathbf{C}, \sqsubseteq)$ *is confluent.*

Proof: For the *if* part it suffices to amend the induction proof of Lemma 5.1.4 by considering formulae of form $\Diamond M$. Assume $c, b \in \mathbf{C}$, $c \sqsubseteq b$, and $c \models \Diamond M$. We wish to prove $b \models \Diamond M$. From $c \models \Diamond M$ we get a, $c \sqsubseteq a$ with $a \models M$. Since $\sqsubseteq$ confluent we know there exists a d such that both $a \sqsubseteq d$ and $b \sqsubseteq d$. The induction hypothesis now yields $d \models M$, whence $b \models \Diamond M$.

For the *only-if* part of the statement take any closed proposition ϕ of $\mathcal{B}$ and put $M \equiv \iota\phi$. Suppose given three constraints a, b, c such that $c \sqsubseteq a$ and $c \sqsubseteq b$. Define a valuation $V_{a,\phi}$ as follows

$$V_{a,\phi}(\xi) \;=_{df}\; \{x \in \mathbf{C} \mid a \sqsubseteq x \;\&\; \phi \vdash \xi\}.$$

This map satisfies the conditions of valuations, and that it has the property

$$x \in V_{a,\phi}(\phi) \;\Leftrightarrow\; a \sqsubseteq x.$$

We are going to exploit the assumption that the $V(M)$ are upper closed on the particular model $(\mathbf{C}, \sqsubseteq, V_{a,\phi}(\phi))$. More precisely, we use that $V_{a,\phi}(\Diamond\iota\phi)$ is upper closed. Now, since $c \sqsubseteq a$ and $a \in V_{a,\phi}(\phi)$ we have $c \in V_{a,\phi}(\Diamond\iota\phi)$. This set is upper closed, so $b \in V_{a,\phi}(\Diamond\iota\phi)$, whence there must be a d, $b \sqsubseteq d$ such that $d \in V_{a,\phi}(\iota\phi) = V_{a,\phi}(\phi)$. The latter is equivalent by construction to $a \sqsubseteq d$. Thus we have found a d that is both above a and b, which completes the proof that $\sqsubseteq$ is confluent. ∎

The following theorem sums up our analysis. It is the central result of this section.

Theorem 5.1.9 (Soundness) *Let $\mathcal{M}$ be the class of confluent constraint models. Then, the rules $\Diamond F$ and $\supset I$ are valid (without restriction) in $\mathcal{M}$. Consequently, all derivable sequents in $\mathcal{L}_0$ are valid in $\mathcal{M}$. In particular, if for some formula M, $\vdash M$ is derivable, then $\mathcal{C} \models M$ for all $\mathcal{C}$ in $\mathcal{M}$.*

Proof: Rule $\supset I$ is valid in $\mathcal{M}$ by Lemmata 5.1.6 and 5.1.8. Rule $\Diamond F$ in its general form can be derived from sequent $\Diamond F^0$, rules $\supset I$, $\supset E$, and the structural rules. Since $\Diamond F^0$, $\supset E$ and the structural rules are valid in every constraint model, $\Diamond F$ must be valid in $\mathcal{M}$. Together with Lemma 5.1.5 we find that all rules of $\mathcal{L}_0$ are valid in $\mathcal{M}$. ∎

Here are a few concrete examples of confluent constraint models:

Example

- Take as constraints C the finite lists $[\gamma_n, \ldots, \gamma_1]$ of well-formed closed propositions and let $[\gamma_n, \ldots, \gamma_1] \sqsubseteq [\delta_m, \ldots, \delta_1]$ if all δ_j occur among the γ_i. For $c = [\gamma_n, \ldots, \gamma_1]$ define

  $$[\gamma_n, \ldots, \gamma_1] \in V(\phi) \quad \Leftrightarrow \quad \gamma_1, \ldots, \gamma_n \vdash \phi$$

  These data constitute a confluent constraint model, $\mathcal{C}_*$.

- A trivial confluent constraint model $\mathcal{C}_0$ is given by the one-element preordered set $\mathrm{C} = \{*\}$, $* \sqsubseteq *$, with constraint valuation $V(\phi) = \{x \mid x = * \;\&\; \vdash \phi\}$, *i.e.* $V(\phi)$ is the singleton set $\{*\}$ if ϕ provable in $\mathcal{B}$ and the empty set otherwise. In this model $\Diamond M$ is semantically equivalent to M.

- Another simple example, name it $\mathcal{C}_c$, is in a way between $\mathcal{C}_*$ and $\mathcal{C}_0$: Take $\mathrm{C} = \{0, 1\}$, with natural ordering $0 \sqsubseteq 0$, $0 \sqsubseteq 1$, and $1 \sqsubseteq 1$ and valuation

  $$0 \in V(\phi) \Leftrightarrow \vdash \phi \qquad 1 \in V(\phi) \Leftrightarrow u \vdash \phi$$

  where u is some fixed proposition of $\mathcal{B}$. This model is linearly ordered and hence confluent. This model $\mathcal{C}_c$ is inspired by Curry [Cur52]).

- Let $(C, 1, \cdot)$ with action $c, \phi \mapsto \phi^c$ be a notion of constraint for $\mathcal{B}$ in the sense defined in Chapter 3, Definition 3.1.9. Then, $(C, \sqsubseteq, V)$ with $c \sqsubseteq d$ *iff* for all ϕ, $\phi^c \vdash \phi^d$ and $V(\phi) = \{c \mid \vdash \phi^c\}$ is a constraint model. Clearly, $\sqsubseteq$ is a preorder; also, $a \sqsubseteq a \cdot b$ and $b \sqsubseteq a \cdot b$, whence $\sqsubseteq$ is confluent.

∎

The discussion in this section confirms that it is possible to give a non-trivial Kripke semantics of $\Diamond$ for which the rules of $\mathcal{L}_0$ are sound. It has been shown that this naturally leads to interpret $\Diamond$ *intuitionistically* on a *confluent* Kripke frame.

We point out that Definitions 5.1.2, 5.1.3 are not the only possible way to give a sound Kripke semantics to $\mathcal{L}_0$. For example, it is not mandatory to use a single frame relation $\sqsubseteq$ to interpret *both* modality $\Diamond$ *and* intuitionistic implication $\supset$. A drastic consequence of being thrifty in this way is that every formula $\Diamond M$ is semantically equivalent to the formula $\neg\neg M$, where $\neg$ abbreviates $(\cdot) \supset \mathit{false}$. In fact, one can verify immediately by checking definitions that

$$\models \Diamond M \cong \neg\neg M$$

This means that the semantics of $\Diamond$ is not independent from the connectives $\supset, \mathit{false}$ and that it may be too strong to be completely captured by derivability in $\mathcal{L}_0$. In fact, $\mathcal{L}_0$ is incomplete:

Theorem 5.1.10 (Incompleteness) *$\mathcal{L}_0$ is incomplete wrt. the Kripke semantics given by Definitions 5.1.2 and 5.1.3. There is a formula N with $C \models N$ for all constraint models C such that $\vdash N$ cannot be derived in $\mathcal{L}_0$.*

Proof: Take the formula $N \equiv \Diamond \mathit{false} \supset \mathit{false}$. Then, $\models N$ but not $\vdash N$. The first claim follows by definition of validity. The second claim is a consequence of Theorem 3.1.19: If the sequent $\vdash \Diamond \mathit{false} \supset \mathit{false}$ were derivable in $\mathcal{L}_0$, then, we must have a derivation of $\vdash \mathit{true} \supset \mathit{false}$ in $\mathcal{L}$. Contradiction to the consistency of $\mathcal{L}$. ∎

5.2 A Category Theoretical Interpretation of Lax Logic

In this section we present a specific category theoretic interpretation of lax logic. It provides a semantical reconstruction of the syntactic process of constraint extraction defined in Chapter 3.2, which, for a given notion of constraint, identifies proofs of $\Diamond\, \iota\, \phi$ with constraints c such that ϕ^c. This interpretation can be characterized by the identification

$$\Diamond\, \iota\, \phi \cong \Sigma c.\, \phi^c$$

where Σ stands for a strong existential quantifier, *i.e.* one that satisfies, for instance, choice axioms like

$$\forall x.\, \Sigma c.\, \phi^c \supset \exists f.\, \forall x.\, \phi^{f\, x}$$

$$(\Sigma c.\, \phi^c \supset \Sigma c.\, \psi^c) \supset \Sigma f.\, \forall c.\, \phi^c \supset \psi^{f\, c}$$

which reflect some properties of constraint extraction. Consequently, the reconstruction we are heading for essentially is a systematic way of extending the base logic by a strong existential quantifier.

The category theoretical construction of lax logic presented in this section is carried out in the shape of a *hyperdoctrine* [Law69,See83,Pit89]. More specifically, it is shown by categorical construction how any base logic $\mathcal{B}$ given as a hyperdoctrine structure can be extended by first-order primitives, in particular a *strong* existential quantifier Σ. In this extension $\mathcal{B}$ is contained via an embedding ι that preserves and reflects provability, so that the new logic is a conservative first-order extension of $\mathcal{B}$.

5.2.1 Base Logic as an Indexed Preorder

We begin with transforming the base logic $\mathcal{B}$ into an indexed preorder

$$\mathcal{B} = (\mathcal{T}, \mathbf{B} : \mathcal{T}^{op} \to \mathbf{PreOrd})$$

where $\mathcal{T}$ is a category and $\mathbf{B}$ a contravariant (pseudo-) functor from $\mathcal{T}$ into the category **PreOrd** of preordered sets and order-preserving maps. Note, we are using the same name $\mathcal{B}$ for both the previously introduced syntactic calculus of the base logic and the indexed category to be constructed below. The translation will follow closely the traditional Lawvere-style approach of representing predicate logical theories in category theoretical terms [Law69]. More specifically, category $\mathcal{T}$ is the categorical representation of the object language of the base logic obtained according to the principles

$$\begin{array}{rclrcl} \textit{objects} & = & \textit{types} & \textit{morphisms} & = & \textit{terms} \\ \textit{composition} & = & \textit{substitution} & \textit{identities} & = & \textit{variables} \end{array}$$

and the indexed (pseudo-) functor $\mathbf{B} : \mathcal{T}^{op} \rightarrow \mathbf{PreOrd}$ representing the logical calculus of the base logic is obtained from the principles

$$\begin{array}{rcl} \textit{elements of fibres} & = & \textit{propositions with free variables} \\ \textit{ordering of fibres} & = & \textit{sequents} \\ \textit{translation maps between fibres} & = & \textit{substitution} \end{array}$$

where with *fibres* we mean the preordered sets $\mathbf{B}[\tau]$, τ object in $\mathcal{T}$. The fact that we do not assume that proofs or derivations in the base logic carry information manifests itself in the fibres being preordered sets rather than arbitrary categories. Most constructions to follow below, we believe generalize to the case where the fibres are proper categories.

Notation: If $\Delta = x_1^{\tau_1}, \ldots, x_n^{\tau_n}$ is a context, then we denote the sequence $\tau_1, \ldots, \tau_n$ of types by $\|\Delta\|$. In the special case where Δ is the empty context $()$, $\|\Delta\|$ is the empty sequence $()$. Given a well-formed term t with free variables in Δ and sequence $s = s_1, \ldots s_n$ of well-formed terms of types $\tau_1, \ldots, \tau_n$, we use the notation $t\{s/\Delta\}$ to abbreviate the substitution $t\{s_1/x_1\} \cdots \{s_n/x_n\}$.

Definition 5.2.1 *The indexed preorder $\mathcal{B} = (\mathcal{T}, \mathbf{B} : \mathcal{T}^{op} \rightarrow \mathbf{PreOrd})$ is given by the following data*

- *$\mathcal{T}$ is the category with* objects *the finite, possibly empty sequences $\tau = \tau_1, \ldots, \tau_n$ of types of the base logic; a* morphism *$\sigma \to \tau$, where $\sigma = \sigma_1, \ldots, \sigma_m$ and $\tau = \tau_1, \ldots \tau_n$, is an equivalence class $[(\Delta, t)]$ of a pair, where the first component Δ is a context with $\|\Delta\| = \sigma$ and the second is a possibly empty sequence $t = t_1, \ldots, t_n$ of well-formed terms of types $\tau_1, \ldots, \tau_n$, respectively, each of which has its free variables in Δ, i.e.*

$$\Delta \vdash t_i : \tau_i \qquad i = 1, \ldots, n,$$

under the equivalence that identifies (Δ, t) with (Δ', t') if t and t' are α-convertible, i.e. $t'\{\Delta/\Delta'\}$ is syntactically identical to t modulo renaming of bound variables.

The identity *morphism $id : \tau \to \tau$ is the equivalence class $[(\Delta, \Delta)]$ where Δ is an arbitrary context such that $\|\Delta\| = \tau$.* Composition *of a morphism $[(\Delta_s, s)] : \rho \to \sigma$, $s = s_1, \ldots, s_m$ with morphism $[\Delta_t, t] : \sigma \to \tau$, $t = t_1, \ldots, t_n$ and $\Delta_t = y_1^{\sigma_1}, \ldots, y_m^{\sigma_m}$ is the equivalence class $[(\Delta_s, t \circ s)] : \rho \to \tau$ where $t \circ s$ is the sequence*

$$t_1\{s/\Delta_t\} \ , \ \ldots \ , \ t_n\{s/\Delta_t\}.$$

- *Given an object $\tau = \tau_1, \ldots, \tau_n$ in $\mathcal{T}$, then the* fibre $\mathbf{B}[\tau]$ *over τ is the preordered set with* elements *the equivalence classes $[(\Delta, \phi)]$ of pairs, where the first component Δ is a context with $\|\Delta\| = \tau$ and the second ϕ a well-formed proposition with free variables in Δ, i.e.*

$$\Delta \vdash \phi : \Omega,$$

under the equivalence that identifies $[(\Delta, \phi)]$ and $[(\Delta', \phi')]$ if ϕ and ϕ' are α-convertible, i.e. $\phi'\{\Delta/\Delta'\}$ is syntactically identical to ϕ modulo renaming of bound variables. Let Δ, Δ' be contexts such that $\|\Delta\| = \|\Delta'\| = \tau$. For elements $[(\Delta, \phi)]$ and $[(\Delta', \phi')]$ in $\mathbf{B}[\tau]$, the ordering $[(\Delta, \phi)] \sqsubseteq [(\Delta', \phi')]$ *is defined to hold iff the sequent*

$$\phi \vdash_\Delta \phi'\{\Delta/\Delta'\}$$

is derivable in the base logic. Finally, given a morphism $[(\Delta', t)] : \tau' \to \tau$ *in* $\mathcal{T}$*, the order-preserving* translation $\mathbf{B}[[(\Delta', t)]] : \mathbf{B}[\tau] \to \mathbf{B}[\tau']$ *is given by the map*

$$[(\Delta, \phi)] \mapsto [(\Delta', \phi\{t/\Delta\})].$$

Notation: Morphisms $[(\Delta, t)]$ in $\mathcal{T}$ will be denoted more suggestively by $[\Delta \vdash t]$, and elements $[(\Delta, \phi)]$ in $\mathbf{B}[\|\Delta\|]$ by $[\Delta \vdash \phi]$ or $[\Delta \vdash \phi : \Omega]$.

Remark: It will simplify later definitions considerably to use the fact that any two morphisms $\sigma \to \tau$ in $\mathcal{T}$ and also any two elements in $\mathbf{B}[\sigma]$ can be normalized to have a common context Δ, $\|\Delta\| = \sigma$, as their free variables. The first example of such use is Lemma 5.2.2 below.

The indexed preorder $\mathcal{B}$ constructed in Definition 5.2.1 is well-defined. Moreover, without assuming any particular properties of the object language, $\mathcal{T}$ has finite products.

Lemma 5.2.2 *The category* $\mathcal{T}$ *has finite products. More specifically, the empty sequence* $()$ *of types is terminal object, and the sequence* σ, τ *is a product of objects* σ *and* τ *in* $\mathcal{T}$*; its projections are* $\pi_1 = [\Delta, \Delta' \vdash \Delta]$ *and* $\pi_2 = [\Delta, \Delta' \vdash \Delta']$ *where* Δ, Δ' *are arbitrary contexts such that* $\|\Delta\| = \sigma$ *and* $\|\Delta'\| = \tau$*. The pairing of morphisms* $[\Delta \vdash s] : \sigma \to \tau_1$ *with* $[\Delta \vdash t] : \sigma \to \tau_2$ *is*

$$\langle s, t \rangle =_{df} [\Delta \vdash s, t] \; : \; \sigma \to \tau_1, \tau_2$$

Proof: easy ∎

Definition 5.2.3 *Let* $\mathcal{I} = (\mathcal{C}, \mathbf{I} : \mathcal{C}^{op} \to \mathbf{PreOrd})$ *be an indexed preorder. A* congruence $\equiv$ *on* $\mathcal{I}$ *is given by two families* $\equiv_{a,b}$ *and* $\equiv_a$ *of relations indexed in objects* a, b *of* $\mathcal{C}$ *such that*

- $\equiv_{a,b}$ *is an equivalence relation on the morphisms in* $\mathcal{C}$ *with domain* a *and codomain* b *and* $\equiv_a$ *is an equivalence relation on the elements of the fibre* $\mathbf{I}[a]$

- $\equiv_{a,b}$ *is preserved by composition of morphisms and* $\equiv_a$ *respects the ordering, i.e. if* $f \equiv_{a,b} f'$ *and* $g \equiv_{b,c} g'$ *then* $g \circ f \equiv_{a,c} g' \circ f'$, *and whenever* $\phi \equiv_a \phi'$, $\psi \equiv_a \psi'$, *and* $\phi \sqsubseteq \psi$, *then* $\phi' \sqsubseteq \psi'$

- *if* $f \equiv_{a,b} f'$, *then for all objects* ϕ, ϕ' *in* $\mathbf{I}[b]$, *if* $\phi \equiv_b \phi'$, *then* $\mathbf{I}[f]\phi \equiv_a \mathbf{I}[f']\phi'$

For a congruence $\equiv$ *on* $\mathcal{I}$ *the induced* quotient *of* $\mathcal{I}$,

$$\mathcal{I}_{\equiv} \;=\; (\mathcal{C}_{\equiv}, \mathbf{I}_{\equiv} : (\mathcal{C}_{\equiv})^{op} \to \mathbf{PreOrd})$$

is defined as follows: $\mathcal{C}_{\equiv}$ *has as objects the objects of* $\mathcal{C}$ *and as morphisms* $a \to b$ *the equivalence classes modulo* $\equiv_{a,b}$ *of morphisms* $f : a \to b$ *in* $\mathcal{C}$, *denoted by* $[f] = [f]_{\equiv_{a,b}}$. *Composition and identities in* $\mathcal{C}_{\equiv}$ *are defined qua representatives, i.e.* $id_a = [id_a]$ *and* $[g] \circ [f] = [g \circ f]$. *The fibres* $\mathbf{I}_{\equiv}[a]$ *have as elements equivalence classes modulo* $\equiv_a$ *of elements* ϕ *in* $\mathbf{I}[a]$, *denoted by* $[\phi] = [\phi]_{\equiv_a}$. *The ordering on* $\mathbf{I}[a]$ *is given qua representatives, i.e.* $[\phi] \sqsubseteq [\psi]$ *iff* $\phi \sqsubseteq \psi$. *Finally, given a morphism* $[f] : a \to b$ *in* $\mathcal{C}_{\equiv}$ *the translation* $\mathbf{I}_{\equiv}[[f]] : \mathbf{I}_{\equiv}[b] \to \mathbf{I}_{\equiv}[a]$ *is the mapping* $[\phi] \mapsto [\mathbf{I}[f]\phi]$.

Remark: It is easy to check that the constructed quotient $\mathcal{I}_{\equiv}$ is a well-defined indexed preorder, in particular that the translation maps $\mathbf{I}_{\equiv}[[f]]$ are independent of the particular choice of the representative f. The notion of a congruence relation on a category and the induced quotient category is standard. It is extended here to work for indexed preorders, but only to the extent that it be sufficient for our purposes. It is not claimed that this definition is helpful in its own right from a category theoretic point of view.

Now we observe that 'provable equality' ($=$) in the base logic can be used to define a congruence on the indexed preorder $\mathcal{B} = (\mathcal{T}, \mathbf{B} : \mathcal{T}^{op} \to \mathbf{PreOrd})$.

Lemma 5.2.4 *Let $\mathcal{B} = (\mathcal{T}, \mathbf{B} : \mathcal{T}^{op} \to \mathbf{PreOrd})$ be the indexed preorder constructed in Definition 5.2.1. For σ and τ objects in $\mathcal{T}$ define a relation $=_{\sigma,\tau}$ on the morphisms $\sigma \to \tau$ in $\mathcal{T}$ by*

$$[\Delta \vdash s] =_{\sigma,\tau} [\Delta \vdash t] \quad \textit{iff} \quad \vdash_\Delta s_i = t_i$$

is derivable in $\mathcal{B}$ for all $i = 1, \ldots, n$ where $s = s_1, \ldots, s_n$ and $t = t_1, \ldots, t_n$. Further, define a relation $=_\tau$ on the elements in $\mathbf{B}[\tau]$ by

$$[\Delta \vdash \phi] =_\tau [\Delta \vdash \psi] \quad \textit{iff} \quad \vdash_\Delta \phi = \psi$$

is derivable in $\mathcal{B}$. Then, $=$ is a congruence on the indexed preorder $\mathcal{B}$.

Proof: easy, by the rules of $\mathcal{B}$. ∎

The indexed preorder $\mathcal{B}$, apart from α-conversion, is a purely syntactic object. However, passing to the quotient $\mathcal{B}_=$ builds in additional identities that turn the base $\mathcal{T}$ into a bicartesian[1] closed category. In order to define this categorical structure we replace $\mathcal{T}_=$ by the full sub-category $\mathcal{T}_0$ generated by single element lists. This will be justified by the fact that in $\mathcal{T}_=$ an object $\tau_1, \ldots, \tau_n$ is isomorphic to $\tau_1 \times \cdots \times \tau_n$.

Notation: In $\mathcal{B}_=$ we have built two equivalence relations on top of each other. Morphisms in $\mathcal{T}_=$ and elements in $\mathbf{B}_=[\sigma]$ should be written $[[\Delta \vdash s]]$, where the inner square brackets stand for a change of context and renaming of bound variable (*i.e.* α *conversion*) and the outer for provable equivalence. In the sequel both brackets will be merged into one, writing $[\Delta \vdash s]$.

Lemma 5.2.5 *Let $\mathcal{T}_=$ be the quotient induced by the congruence of Lemma 5.2.4 on $\mathcal{T}$. Then, the full sub-category $\mathcal{T}_0$ of $\mathcal{T}_=$ generated by single element lists τ is equivalent to $\mathcal{T}_=$.*

[1]The term *bicartesian* here and in the following means 'cartesian closed and equipped with finite coproducts'

Proof: An equivalence between categories $\mathcal{T}_0$ and $\mathcal{T}_=$ is a pair of functors $F : \mathcal{T}_0 \to \mathcal{T}_=$ and $G : \mathcal{T}_= \to \mathcal{T}_0$ together with natural isomorphisms $FG \cong 1 : \mathcal{T}_= \to \mathcal{T}_=$ and $GF \cong 1 : \mathcal{T}_0 \to \mathcal{T}_0$. Now $\mathcal{T}_0$ is full sub-category of $\mathcal{T}_=$, so for F we take the inclusion functor which is full and faithful. The existence of G and the natural isomorphisms, then, is equivalent to the condition that each object $\tau = \tau_1, \ldots, \tau_n$ in $\mathcal{T}_=$ is isomorphic to some object in $\mathcal{T}_0$. One can show that such an object is $\tau_1 \times \cdots \times \tau_n$, where it is understood, say, that $\times$ associates to the right. As one half of the isomorphism $\tau \to \tau_1 \times \cdots \times \tau_n$ in $\mathcal{T}_=$ take the equivalence class $[\Delta \vdash (x_1, (x_2, (\ldots x_n) \cdots))]$ where $\Delta = x_1^{\tau_1}, \ldots, x_n^{\tau_n}$ is a context, and for the other direction $\tau_1 \times \cdots \times \tau_n \to \tau$ the equivalence class $[z \vdash \pi_1^n\, z, \ldots, \pi_n^n\, z]$, where z is a variable of type $\tau_1 \times \cdots \times \tau_n$ and π_k^n is the k-th projection $\tau_1 \times \cdots \times \tau_n \to \tau_k$, $k = 1, \ldots, n$. Formally, $\pi_k^n\, z$ for $1 \leq k \leq n$ can be defined inductively by

$$\pi_{k+1}^{n+1}\, z = \pi_k^n\, (\pi_2\, z) \qquad \pi_1^{n+1}\, z = \pi_1\, z \qquad \pi_1^1\, z = z.$$

That both maps are mutually inverse is due to equivalence classes being taken modulo provable equality. Note, the equivalence is independent of the choices of variables made, and it covers the case $n = 1$. The case $n = 0$, *i.e.* $\tau = ()$ is treated separately: the corresponding object in $\mathcal{T}_0$ is $\mathbf{1}$. One half of the isomorphism $() \to \mathbf{1}$ is given by $[\vdash *]$ and the other half $\mathbf{1} \to ()$ by $[z \vdash]$, where z is a variable of type $\mathbf{1}$. ∎

Now we identify the promised bicartesian structure in $\mathcal{T}_0$. It is induced by the type constructors $\mathbf{1}, \mathbf{0}, \times, +, \Rightarrow$, the associated operations on terms (constructors and destructors), and equations provable in the base logic.

Lemma 5.2.6 *Let category $\mathcal{T}_0$ be as in Lemma 5.2.5. Then, $\mathcal{T}_0$ is bicartesian closed.*

Proof: We will only point to the relevant categorical data and omit proving their universal properties, which comes down to proving certain equations in the base logic. The construction of syntactic categories from lambda-calculi is

well-known, see *e.g.* [LS86]. Recall that objects in $\mathcal{T}_0$ are (single) types σ and morphisms $\sigma \to \tau$ are equivalence classes $[z \vdash t]$ of well-formed terms t of type τ with (at most) a single variable z of type σ as its free variable. Two such equivalence classes $[z \vdash t]$ and $[z \vdash t']$ are identical *iff* $\vdash_z t = t'$ is derivable in $\mathcal{B}$.

- Terminal object is the type $\mathbf{1}$. Given an object τ the unique morphism $!_\tau : \tau \to \mathbf{1}$ is the class $[z \vdash *]$ where z is any variable of type τ.

- The product of objects σ and τ is $\sigma \times \tau$. The first projection $\sigma \times \tau \to \sigma$ is $[z \vdash \pi_1 z]$, the second $\sigma \times \tau \to \tau$ is $[z \vdash \pi_2 z]$ where z is any variable of type $\sigma \times \tau$. Given morphisms $[x \vdash f] : \alpha \to \sigma$ and $[x \vdash g] : \alpha \to \tau$, their pairing is $[x \vdash (f, g)] : \alpha \to \sigma \times \tau$. The class $[x \vdash (f, g)]$ is invariant under change of representatives f, g and variable x.

- Initial object is the type $\mathbf{0}$. Given an object τ the unique morphism $\Box_\tau : \mathbf{0} \to \tau$ is the class $[z \vdash \Box z]$ where z is any variable of type $\mathbf{0}$.

- The coproduct of objects σ and τ is the type $\sigma + \tau$. The first injection $\sigma \to \sigma + \tau$ is $[z \vdash \iota_1 z]$ with z variable of type σ, the second injection $\tau \to \sigma + \tau$ is $[z \vdash \iota_2 z]$ with z variable of type τ. Given morphisms $[x \vdash f] : \sigma \to \alpha$ and $[y \vdash g] : \tau \to \alpha$, their sum is

$$[z \vdash case_{x,y}(z, f, g)] : \sigma + \tau \to \alpha$$

  where z is any variable of type $\sigma + \tau$. The class $[z \vdash case_{x,y}(z, f, g)]$ is invariant under change of representatives f, g and variable z.

- The exponent of objects σ and τ is the type $\sigma \Rightarrow \tau$. Evaluation is the class

$$[z \vdash (\pi_2 z)(\pi_1 z)] : \sigma \times (\sigma \Rightarrow \tau) \to \tau$$

  where z is any variable of type $\sigma \times (\sigma \Rightarrow \tau)$. For a morphism $[z \vdash t] : (\alpha \times \sigma) \to \tau$ define its currying as

$$[x \vdash \lambda y.\, t\{(x, y)/z\}] : \alpha \to (\sigma \Rightarrow \tau)$$

where x, y arbitrary variables of type α, σ, respectively. The equivalence class $[x \vdash \lambda y.\, t\{(x,y)/z\}]$ does not depend on the particular choice of representative t and variables x, y.

∎

Remark: The lemma only states that the required data exists but the proof actually defines a canonical choice of categorical products, coproducts, *etc.* Note, the constructions are independent of the choices for variables named 'z' — or 'x', 'y' in the case of exponents — so they need not be picked in a canonical way.

Corollary 5.2.7 *Let $\mathcal{T}_=$ be as in Lemma 5.2.5. Then, $\mathcal{T}_=$ is a bicartesian closed category.*

Proof: Follows from Lemmata 5.2.5, 5.2.6, and the fact that equivalences between categories preserve limits and colimits. ∎

Remark: Since a canonical choice for the bicartesian structure is picked in $\mathcal{T}_0$ the specific equivalence between $\mathcal{T}_0$ and $\mathcal{T}_=$ constructed in Lemma 5.2.5 can be used to lift this choice to $\mathcal{T}_=$. The lifted structure in $\mathcal{T}_=$ also will be denoted by the symbols $\mathbf{1}, \mathbf{0}, \times, +, \Rightarrow$. Note, that we have got two different finite product structures on $\mathcal{T}_=$: $\mathbf{1}$ and $\sigma \times \tau$ lifted from $\mathcal{T}_0$ as well as $()$ and σ, τ as identified in Lemma 5.2.2 on $\mathcal{T}$ pushed into the quotient $\mathcal{T}_=$ in the obvious way. Both product structures are isomorphic but not identical. Since it will sometimes be important to be clear about which one is meant let us call the first one $\mathcal{T}_0$-products, the latter $\mathcal{T}$-products. In general, the bicartesian structure induced by $\mathcal{T}_0$ will be referred to as the $\mathcal{T}_0$-structure.

We have seen above that the base category $\mathcal{T}_=$ of $\mathcal{B}_=$ is bicartesian closed. Now we are going to investigate some of the structure that is induced on the indexed preorder $\mathcal{B}_=$ by the various properties that we assumed of the syntactic calculus of the base logic in Chapter 3. First recall the definition of a *hyperdoctrine* [Law69,See83], the categorical equivalent of (first-order) predicate logical theories.

Definition 5.2.8 *A* hyperdoctrine *is an indexed category* $\mathcal{I} = (\mathcal{C}, \mathbf{I} : \mathcal{C}^{op} \to \mathbf{Cat})$ *with the following properties*

- $\mathcal{C}$ *has finite products*
- *for each object* a *in* $\mathcal{C}$ *the fibre* $\mathbf{I}[a]$ *is bicartesian closed.*
- *for each morphism* $t : a \to b$ *in* $\mathcal{C}$ *the translation map* $\mathbf{I}[t] : \mathbf{I}[b] \to \mathbf{I}[a]$ *preserves the bicartesian closed structure.*
- *translations along projections have right and left adjoints satisfying the Beck-Chevalley Condition (for certain pullback squares). More precisely, for* $\pi_1 : a \times b \to a$ *first projection in* $\mathcal{C}$ *there are functors*

$$\forall[\pi_1] : \mathbf{I}[a \times b] \to \mathbf{I}[a] \quad \textit{and} \quad \exists[\pi_1] : \mathbf{I}[a \times b] \to \mathbf{I}[a]$$

with $\mathbf{I}[\pi_1] \dashv \forall[\pi_1]$ *and* $\exists[\pi_1] \dashv \mathbf{I}[\pi_1]$. *The Beck-Chevalley Condition holds for pullback squares of the form*

$$\begin{array}{ccc} a \times b & \xrightarrow{\pi_1} & a \\ {\scriptstyle t \times id}\downarrow & & \downarrow{\scriptstyle t} \\ a' \times b & \xrightarrow[\pi_1]{} & a' \end{array}$$

where $t : a \to a'$ *morphism in* $\mathcal{C}$, *i.e. we have* $\mathbf{I}[t] \circ \forall[\pi_1] \cong \forall[\pi_1] \circ \mathbf{I}[t \times id]$ *and* $\mathbf{I}[t] \circ \exists[\pi_1] \cong \exists[\pi_1] \circ \mathbf{I}[t \times id]$.

Remark: This definition of a hyperdoctine is a little less restrictive than the one given in [See83] in that Beck-Chevalley needs to hold only for pull-back squares generated by first projections rather than for all pull-backs as in [See83]. These simplification is adopted for mere convenience and not for technical reasons.

We emphasize that in order to verify the last clause of the definition one may pick an arbitrary product structure $a \times b$ on $\mathcal{C}$ that is chosen canonically for each pair of objects a, b. In particular, it need not be the product of the first clause in the definition. If it is satisfied for one choice, then it holds for any other, too.

Theorem 5.2.9 $\mathcal{B}_= = (\mathcal{T}_=, \mathbf{B}_= : \mathcal{T}_=^{op} \to \mathbf{PreOrd})$ *is a hyperdoctrine with the following additional properties*

- $\mathcal{T}_=$ *is bicartesian closed*
- *there is a natural family of bijections* $[\tau \to \Omega] \cong \mathbf{B}_=[\tau]$ *indexed in objects* $\tau = \tau_1, \ldots, \tau_n$ *of* $\mathcal{T}_=$, *where* $[\tau \to \Omega]$ *denotes the set of all morphisms* $\tau \to \Omega$ *in* $\mathcal{T}_=$.
- *Translations* $\mathbf{B}_=[id \times \iota_1]$ *along morphisms* $id \times \iota_1 : \alpha \times \sigma \to \alpha \times (\sigma + \tau)$ *where* $\iota_1 : \sigma \to \sigma + \tau$ *is a first injection have left adjoints*

$$\exists[id \times \iota_1] : \mathbf{B}_=[\alpha \times \sigma] \to \mathbf{B}_=[\alpha \times (\sigma + \tau)]$$

which satisfy Beck-Chevalley for pull-back squares of the shape

$$\begin{array}{ccc} \alpha \times \sigma & \xrightarrow{t \times id} & \alpha' \times \sigma \\ {\scriptstyle id \times \iota_1}\downarrow & & \downarrow{\scriptstyle id \times \iota_1} \\ \alpha \times (\sigma + \tau) & \xrightarrow[t \times id]{} & \alpha' \times (\sigma + \tau) \end{array}$$

with $t : \alpha \to \alpha'$ *arbitrary morphism in* $\mathcal{T}_=$, *i.e.*

$$\exists[id \times \iota_1]\mathbf{B}_=[t \times id] \cong \mathbf{B}_=[t \times id]\exists[id \times \iota_1].$$

Remark: The additional properties of $\mathcal{B}_=$ mentioned are crucial for our purposes and will be used later. The second of these properties, *viz.* the natural bijection $[\tau \to \Omega] \cong \mathbf{B}_=[\tau]$ reflects the higher order nature of the base logic. The last clause could be strengthened to encompass general quantifiers (*i.e.* left and right adjoints to *arbitrary* translations) but we refrain from doing so since we will only need the particular adjoints $\exists[id \times \iota_1]$. $\mathcal{B}_=$ has other properties not mentioned, like list objects and natural numbers object, which we will not need.

In the following proof of the theorem we will use the $\mathcal{T}$-product and $\mathcal{T}_0$-sum to verify the last condition. But again, this means the condition will be true for

any other canonical choice of products and sums. In fact, we will use the last condition later with the $\mathcal{T}_0$ rather than $\mathcal{T}$-product.

Proof: We begin with verifying that $\mathcal{B}_=$ is a hyperdoctrine. That $\mathcal{T}_=$ has finite products has been proven already. The following will assume the $\mathcal{T}$-products from Lemma 5.2.2 (also *cf.* the remark following Corollary 5.2.7).

It is to be seen that for each $\tau = \tau_1, \ldots, \tau_n$ in $\mathcal{T}_=$ the fibre $\mathbf{B}_=[\tau]$ is bicartesian closed. Recall that the elements in $\mathbf{B}_=[\tau]$ are equivalence classes $[\Delta \vdash \phi]$ with $\|\Delta\| = \tau$ and ϕ well-formed proposition with free variables in Δ, *i.e.* $\Delta \vdash \phi : \Omega$. Two such classes $[\Delta \vdash \phi]$ and $[\Delta \vdash \psi]$ are equal *iff* $\vdash_\Delta \phi = \psi$ is derivable. The ordering in $\mathbf{B}_=[\tau]$ is such that $[\Delta \vdash \phi] \sqsubseteq [\Delta \vdash \psi]$ *iff* $\phi \vdash_\Delta \psi$ is derivable. Finally, if $[\Delta' \vdash t]$, $t = t_1, \ldots, t_n$ is a morphism $\tau' \to \tau$ in $\mathcal{T}_=$, then the translation $\mathbf{B}_=[[\Delta' \vdash t]][\Delta \vdash \phi]$ is the element $[\Delta' \vdash \phi\{t/\Delta\}]$.

Here are the definitions qua representatives that turn $\mathbf{B}_=[\tau]$ into a bicartesian closed category; assuming ϕ, ψ are elements in $\mathbf{B}_=[\tau]$ and Δ context with $\|\Delta\| = \tau$, one puts

$$\begin{aligned}
\mathit{true} &\stackrel{df}{=} [\Delta \vdash \mathit{true}] \\
[\Delta \vdash \phi] \wedge [\Delta \vdash \psi] &\stackrel{df}{=} [\Delta \vdash \phi \wedge \psi] \\
\mathit{false} &\stackrel{df}{=} [\Delta \vdash \mathit{false}] \\
[\Delta \vdash \phi] \vee [\Delta \vdash \psi] &\stackrel{df}{=} [\Delta \vdash \phi \vee \psi] \\
[\Delta \vdash \phi] \supset [\Delta \vdash \psi] &\stackrel{df}{=} [\Delta \vdash \phi \supset \psi]
\end{aligned}$$

where

$$\begin{aligned}
\mathit{true} &\equiv \forall z^\Omega . z \supset z \\
\phi \wedge \psi &\equiv \forall z^\Omega . (\phi \supset (\psi \supset z)) \supset z \\
\mathit{false} &\equiv \forall z^\Omega . z \\
\phi \vee \psi &\equiv \forall z^\Omega . (\phi \supset z) \supset ((\psi \supset z) \supset z)
\end{aligned}$$

The definitions are independent of the choice of representatives ϕ, ψ and context Δ, and variable z which must not occur in Δ. That these operations form a bicartesian closed preorder is immediate by definition of $\sqsubseteq$ and the rules of the

base logic. It is also easy to see that the structure is preserved by translations. It is even preserved *on-the-nose*, for instance

$$\begin{aligned}
&\mathbf{B}_{=}[[\Delta' \vdash t]]([\Delta \vdash \phi] \vee [\Delta \vdash \psi]) \\
&\quad = \mathbf{B}_{=}[[\Delta' \vdash t]]([\Delta \vdash \phi \vee \psi]) \\
&\quad = [\Delta' \vdash (\phi \vee \psi)\{t/\Delta\}] \\
&\quad = [\Delta' \vdash \phi\{t/\Delta\} \vee \psi\{t/\Delta\}] \\
&\quad = [\Delta' \vdash \phi\{t/\Delta\}] \vee [\Delta' \vdash \psi\{t/\Delta\}] \\
&\quad = (\mathbf{B}_{=}[[\Delta' \vdash t]][\Delta \vdash \phi]) \vee (\mathbf{B}_{=}[[\Delta' \vdash t]][\Delta \vdash \psi]).
\end{aligned}$$

Let $\pi_1 : \tau \times \tau' \to \tau$ be a first projection, *i.e.* $\tau \times \tau' = \tau, \tau'$ and $\pi_1 = [\Delta, \Delta' \vdash \Delta]$, with Δ, Δ' contexts such that $\|\Delta\| = \tau$ and $\|\Delta'\| = \tau'$. We need to define right and left adjoints $\forall[\pi_1], \exists[\pi_1]$ of translations $\mathbf{B}_{=}[\pi_1] : \mathbf{B}_{=}[\tau] \to \mathbf{B}_{=}[\tau, \tau']$. Given $[\Delta, \Delta' \vdash \phi]$ in $\mathbf{B}_{=}[\tau, \tau']$ and $\Delta' = x_1, \ldots, x_n$, these are obtained by

$$\begin{aligned}
\forall[\pi_1][\Delta, \Delta' \vdash \phi] &= [\Delta \vdash \forall x_1. \cdots \forall x_n. \phi] \\
\exists[\pi_1][\Delta, \Delta' \vdash \phi] &= [\Delta \vdash \exists x_1. \cdots \exists x_n. \phi]
\end{aligned}$$

where $\exists x.\, A \equiv \forall z^{\Omega}.\, (\forall x.\, (A \supset z)) \supset z$. Again, these definitions are independent of the choice of representative ϕ (due to ξ-equality for $\forall$. Recall that $\exists$ in $\mathcal{B}$ is defined via $\forall$.), contexts Δ, Δ', and variable z. $\forall[\pi_1]$ and $\exists[\pi_1]$ are right and left adjoints to $\mathbf{B}_{=}[\pi_1]$, and both satisfy Beck-Chevalley in the strong sense: For a pull-back diagram

$$\begin{array}{ccc}
\tau_1 \times \tau_2 & \xrightarrow{\pi_1} & \tau_1 \\
{\scriptstyle [\Delta_1 \vdash t] \times [\Delta_2 \vdash \Delta_2]}\downarrow & & \downarrow{\scriptstyle [\Delta_1 \vdash t]} \\
\tau_1' \times \tau_2 & \xrightarrow[\pi_1]{} & \tau_1'
\end{array}$$

with $\Delta_2 = x_1, \ldots, x_n$ one obtains

$$\begin{aligned}
&\exists[\pi_1]\mathbf{B}_{=}[[\Delta_1 \vdash t] \times [\Delta_2 \vdash \Delta_2]][\Delta_1', \Delta_2 \vdash \phi] \\
&\quad = \exists[\pi_1]\mathbf{B}_{=}[[\Delta_1, \Delta_2 \vdash t, \Delta_2]][\Delta_1', \Delta_2 \vdash \phi]
\end{aligned}$$

$$
\begin{aligned}
&= \exists[\pi_1][\Delta_1, \Delta_2 \vdash \phi\{t, \Delta_2/\Delta_1', \Delta_2\}] \\
&= \exists[\pi_1][\Delta_1, \Delta_2 \vdash \phi\{t/\Delta_1'\}] \\
&= [\Delta_1 \vdash \exists x_1.\cdots\exists x_n.(\phi\{t/\Delta_1'\})] \\
&= [\Delta_1 \vdash (\exists x_1.\cdots\exists x_n.\phi)\{t/\Delta_1'\}] \\
&= \mathbf{B}_{=}[[\Delta_1 \vdash t]]\exists[\pi_1][\Delta_1', \Delta_2 \vdash \phi]
\end{aligned}
$$

and similarly for $\forall[\pi_1]$. This completes the proof that $\mathcal{B}_{=}$ is a hyperdoctrine.

Next, the additional properties of $\mathcal{B}_{=}$ will be dealt with. By Corollary 5.2.7, $\mathcal{T}_{=}$ is bicartesian closed. That verifies the first property.

For the second property it is to be shown that there is a natural family of bijections $[\tau \to \Omega] \cong \mathbf{B}_{=}[\tau]$ indexed in objects τ of $\mathcal{T}_{=}$. This bijection, however, is trivial since by definition the two sides of $\cong$ are coextensional, *viz.* both have as elements the equivalence classes $[\Delta \vdash t]$ of well-formed terms of type Ω with free variables in Δ such that $\|\Delta\| = \tau$. Naturality amounts to proving that the diagram

$$
\begin{array}{ccc}
[\tau \to \Omega] & \cong & \mathbf{B}_{=}[\tau] \\
{\scriptstyle h \mapsto h \circ [\Delta \vdash t]}\uparrow & & \uparrow{\scriptstyle \mathbf{B}_{=}[[\Delta \vdash t]]} \\
[\tau' \to \Omega] & \cong & \mathbf{B}_{=}[\tau']
\end{array}
$$

commutes for all $[\Delta \vdash t] : \tau \to \tau'$ in $\mathcal{T}_{=}$. But this is trivial again since both maps $h \mapsto h \circ [\Delta \vdash t]$ and $\mathbf{B}_{=}[[\Delta \vdash t]]$ are coextensional by definition.

Finally, we need left adjoints for translation along $id \times \iota_1 : \alpha \times \sigma \to \alpha \times (\sigma + \tau)$ where $\iota_1 : \sigma \to \sigma + \tau$ is a first injection. Here α, σ, τ are objects in $\mathcal{T}_{=}$, *i.e.* $\alpha = \alpha_1, \ldots, \alpha_k$, $\sigma = \sigma_1, \ldots, \sigma_l$, and $\tau = \tau_1, \ldots, \tau_m$. Further, as the coproduct $\sigma + \tau$ of σ and τ we take the $\mathcal{T}_0$-sum, *i.e.* $\sigma + \tau$ is the single-element type sequence $(\sigma_1 \times \cdots \times \sigma_l) + (\tau_1 \times \cdots \times \tau_m)$, and

$$\iota_1 = [\Delta_\sigma \vdash \iota_1(x_1, (\ldots, x_l)\cdots) : \sigma + \tau]$$

is the first injection, where $\|\Delta_\sigma\| = \|x_1, \ldots, x_l\| = \sigma$. Since the shape of the term $\iota_1(x_1, (\ldots, x_l)\cdots)$ will not matter in the following it is abbreviated by i. The

morphism $id \times \iota_1$, then, is $[\Delta_\alpha, \Delta_\sigma \vdash \Delta_\alpha, i]$, where $\|\Delta_\alpha\| = \alpha$ and Δ_α disjoint from Δ_σ.

Given an object $[\Delta_\alpha, \Delta_\sigma \vdash \phi]$ in $\mathbf{B}_=[\alpha \times \sigma]$, we define the object

$$\exists[id \times \iota_1][\Delta_\alpha, \Delta_\sigma \vdash \phi] \stackrel{df}{=} [\Delta_\alpha, z \vdash \exists x_1. \cdots \exists x_l. i = z \wedge \phi]$$

where z is a fresh variable of type $\sigma + \tau$ that does not occur in $\Delta_\alpha, \Delta_\sigma$. The definition is independent of the choice of representative ϕ — because of ξ-equality for $\forall$ — and contexts $\Delta_\alpha, \Delta_\sigma$. That $\exists[id \times \iota_1]$ is left adjoint to $\mathbf{B}_=[id \times \iota_1]$ now follows from the properties of $\exists$ and $=$ in $\mathcal{B}$. It is a functor, *i.e.* monotone function, since

$$\frac{\exists x_1. \cdots \exists x_l. i = z \wedge \phi \vdash_{\Delta_\alpha, z} \exists x_1. \cdots \exists x_l. i = z \wedge \psi}{\phi \vdash_{\Delta_\alpha, \Delta_\sigma} \psi}$$

is a derived rule in $\mathcal{B}$, and it is a left adjoint since

$$\frac{\exists x_1. \cdots \exists x_l. i = z \wedge \phi \vdash_{\Delta_\alpha, z} \psi}{\phi \vdash_{\Delta_\alpha, \Delta_\sigma} \psi\{i/z\}} \qquad \frac{\phi \vdash_{\Delta_\alpha, \Delta_\sigma} \psi\{i/z\}}{\exists x_1. \cdots \exists x_l. i = z \wedge \phi \vdash_{\Delta_\alpha, z} \psi}$$

are derived rules in $\mathcal{B}$. It remains to show that $\exists[id \times \iota_1]$ satisfies Beck-Chevalley. To this end suppose $[\Delta_\alpha \vdash t]$ is a morphism $\alpha \to \alpha'$ and $[\Delta_{\alpha'}, \Delta_\sigma \vdash \phi]$ an object in $\mathbf{B}_=[\alpha' \times \sigma]$. We have to show that

$$\exists[id \times \iota_1]\mathbf{B}_=[t \times id][\Delta_{\alpha'}, \Delta_\sigma \vdash \phi] \cong \mathbf{B}_=[t \times id]\exists[id \times \iota_1][\Delta_{\alpha'}, \Delta_\sigma \vdash \phi]$$

holds in $\mathbf{B}_=[\alpha \times (\sigma + \tau)]$. If both sides are computed using the definitions one obtains

$$\begin{aligned}&[\Delta_\alpha, z \vdash \exists x_1. \cdots \exists x_l. i = z \wedge (\phi\{t/\Delta_{\alpha'}\})] \\ &\quad \cong [\Delta_\alpha, z \vdash (\exists x_1. \cdots \exists x_l. i = z \wedge \phi)\{t/\Delta_{\alpha'}\}]\end{aligned}$$

which is certainly true, in fact, the equivalence is an identity. ∎

5.2.2 Lax Logic as Indexed Category

From the indexed preorder $\mathcal{B}_=$ we now construct a new indexed category

$$\mathcal{DB}_= \;=\; (\mathcal{T}_=, \mathbf{DB}_= : \mathcal{T}_=^{op} \to \mathbf{Cat})$$

in which we plan to interpret lax logic eventually. Its main aspect is that it will provide enough structure to interpret the $\Diamond$ operator. Note, the fibres $\mathbf{DB}[\tau]$ are no longer preordered sets but proper categories, so that now proofs carry non-trivial information. The information we are interested in, of course, is constraint information.

Definition 5.2.10 *Let $\mathcal{C}$ be a category with finite products and $\mathcal{I} = (\mathcal{C}, \mathbf{I} : \mathcal{C}^{op} \to \mathbf{PreOrd})$ be an indexed preorder. The indexed category*

$$\mathcal{DI} = (\mathcal{C}, \mathbf{DI} : \mathcal{C}^{op} \to \mathbf{Cat})$$

is defined as follows: Objects of the fibre $\mathbf{DI}[a]$ for a in $\mathcal{C}$ are pairs (S, ϕ) where S is an object of $\mathcal{C}$ and ϕ object in $\mathbf{I}[a \times S]$. A morphism $(S, \phi) \to (T, \psi)$ in $\mathbf{DI}[a]$ is a morphism $f : a \times S \to T$ in $\mathcal{C}$ such that $\phi \sqsubseteq \mathbf{I}[\langle \pi_1, f \rangle]\psi$. The identity over (S, ϕ) is the morphism $\pi_2 : a \times S \to S$ and composition of $f : (S, \phi) \to (T, \psi)$ and $g : (T, \psi) \to (U, \theta)$ is

$$g \circ \langle \pi_1, f \rangle : a \times S \to U.$$

For each $t : a \to b$ in $\mathcal{C}$ the translation functor $\mathbf{DI}[t] : \mathbf{DI}[b] \to \mathbf{DI}[a]$ is given by the assignment

$$\begin{array}{ccc} (S, \phi) & & (S, \mathbf{I}[t \times id_S]\phi) \\ \Big\downarrow f & \mapsto & \Big\downarrow f \circ (t \times id_S) \\ (T, \psi) & & (T, \mathbf{I}[t \times id_T]\psi) \end{array}$$

Remark: The fibre $\mathbf{DB}[1]$ over the terminal object is precisely the Grothendieck category induced by the indexed category $\mathcal{I}$ (see *e.g.* [BW90]). The above definition of $\mathcal{DB}_=$ is a modification of the standard Grothendieck construction that makes essential use of a the product structure in $\mathcal{C}$. Thus, whenever we talk about $\mathcal{DI}$ for some $\mathcal{I}$ we need to identify, at least implicitly, a particular product structure on the base of $\mathcal{I}$.

To see what is going on let us apply the definition to obtain the indexed category $\mathcal{DB}_= = (\mathcal{T}_=, \mathbf{DB}_= : \mathcal{T}_=^{op} \to \mathbf{Cat})$ using the $\mathcal{T}$-products on $\mathcal{T}_=$, and

reinterpret it in terms of the data of the original base logic $\mathcal{B}$. Recall that objects in $\mathcal{T}_=$ are finite sequences τ of types and a morphism $\tau \to \tau'$ is an equivalence class of sequences of well-formed terms whose number and types are given by τ' and whose free variables are specified by τ. In order to explain functor $\mathbf{DB}_=$ we unroll Definitions 5.2.1, 5.2.10, and the definition of products as in Lemma 5.2.2: The objects in the fibre $\mathbf{DB}_=[\tau]$ are pairs $(\tau', [\Delta, \Delta' \vdash \phi])$ where Δ, Δ' are contexts such that $\|\Delta'\| = \tau'$, $\|\Delta\| = \tau$, and ϕ a well-formed proposition with free variables in Δ, Δ'. Note, $\|\Delta, \Delta'\| = \tau, \tau' = \tau \times \tau'$. A morphism

$$(\tau_1, [\Delta, \Delta_1 \vdash \phi_1]) \to (\tau_2, [\Delta, \Delta_2 \vdash \phi_2])$$

in $\mathbf{DB}_=[\tau]$ is morphism

$$[\Delta, \Delta_1 \vdash t] : \tau \times \tau_1 \to \tau_2$$

in $\mathcal{T}_=$ such that

$$\begin{aligned} [\Delta, \Delta_1 \vdash \phi_1] &\sqsubseteq \mathbf{B}_=[\langle \pi_1, [\Delta, \Delta_1 \vdash t] \rangle][\Delta, \Delta_2 \vdash \phi_2] \\ &= \mathbf{B}_=[[\Delta, \Delta_1 \vdash \Delta, t]][\Delta, \Delta_2 \vdash \phi_2] \\ &= [\Delta, \Delta_1 \vdash \phi_2\{\Delta, t/\Delta, \Delta_2\}] \\ &= [\Delta, \Delta_1 \vdash \phi_2\{t/\Delta_2\}] \end{aligned}$$

which, by definition, means there is a derivation of the $\mathcal{B}$

$$\phi_1 \vdash_{\Delta, \Delta_1} \phi_2\{t/\Delta_2\}.$$

This analysis suggests to view an object $(\tau_2, [\Delta, \Delta_2 \vdash \phi_2])$ as a *specification* of a list of elements of types τ_2 and a morphism $[\Delta, \Delta_1 \vdash t]$ with codomain $(\tau_2, [\Delta, \Delta_2 \vdash \phi_2])$ as an *implementation* that satisfies specification ϕ_2. We remark that the fact that Δ is a context of variables free in both t and ϕ_2 makes sure that ϕ_2 can specify an explicit relationship between input and output.

Remark: Morphisms in $\mathbf{DB}_=[()]$ (the empty sequence of types () is terminal in $\mathcal{T}_=$) are *first-order deliverables* [BM92,McK92]. Thus, the indexed category $\mathcal{DB}_=$

is an indexed category of first-order deliverables (over $\mathcal{B}$) *with free variables.* In [Men91b] the objects in $\mathcal{DB}_{=}[\Delta]$ were called *pointwise designs* and the indexed structure $\mathbf{DB}_{=}$ the *first-order logic of designs.* The possible application of $\mathcal{DB}$ as a *logic* of deliverables , however, is not of concern to us here, so we will not expand on it further. (See Chapter 6 for comparison with McKinna's work.)

Lemma 5.2.11 *Let $\mathcal{C}$, indexed preorder $\mathcal{I} = (\mathcal{C}, \mathbf{I} : \mathcal{C}^{op} \to \mathbf{PreOrd})$, and $\mathcal{DI}$ as in Definition 5.2.10. Then $\mathcal{DI}$ is a conservative extension of $\mathcal{I}$, i.e. there exists a full and faithful embedding*

$$\iota : \mathbf{I} \dot{\to} \mathbf{DI}$$

of indexed categories.

Proof: Fix an object a in $\mathcal{C}$. Let the component $\iota_a : \mathbf{I}[a] \to \mathbf{DI}[a]$ of the natural transformation ι be defined by the assignment

$$\begin{array}{ccc} \phi & & (1, \mathbf{I}[\pi_1]\phi) \\ \sqsubseteq \downarrow & \mapsto & \downarrow !_{a\times 1} \\ \psi & & (1, \mathbf{I}[\pi_1]\psi) \end{array}$$

The short proof that ι_a is a full functor and natural in a is omitted. It is trivially faithful, for the fibres $\mathbf{I}[a]$ are preordered sets. ∎

Remark: The construction of ι depends on the product $\times$ used in the construction of $\mathcal{DI}$ from $\mathcal{I}$. On the other hand, the terminal object $\mathbf{1}$ used in its definition may be any choice of a terminal object.

The main point about the construction of $\mathcal{DI}$ from $\mathcal{I}$ is that in $\mathcal{DI}$ we get (strong) existential quantification for free, independent of whether $\mathcal{I}$ has it or not.

Lemma 5.2.12 *Let $\mathcal{C}$, indexed preorder $\mathcal{I} = (\mathcal{C}, \mathbf{I} : \mathcal{C}^{op} \to \mathbf{PreOrd})$, and $\mathcal{DI} = (\mathcal{C}, \mathbf{DI} : \mathcal{C}^{op} \to \mathbf{Cat})$ be as in Definition 5.2.10. Let $\times$ be the finite products on $\mathcal{C}$. Then, $\mathcal{DI}$ has left adjoints for translations along projections satisfying the*

strong Beck-Chevalley Condition (for certain pullback squares). *More precisely, for $\pi_1 : a \times b \to a$ a projection in $\mathcal{C}$ there is a functor $\Sigma[\pi_1] : \mathbf{DI}[a \times b] \to \mathbf{DI}[a]$ with $\Sigma[\pi_1] \dashv \mathbf{DI}[\pi_1]$. The strong Beck-Chevalley Condition holds for pullback squares of the form*

$$\begin{array}{ccc} a \times b & \xrightarrow{\pi_1} & a \\ {\scriptstyle f \times id}\downarrow & & \downarrow{\scriptstyle f} \\ a' \times b & \xrightarrow[\pi_1]{} & a' \end{array}$$

where $f : a \to a'$ in $\mathcal{C}$, i.e. we have $\mathbf{DI}[f] \circ \Sigma[\pi_1] = \Sigma[\pi_1] \circ \mathbf{DI}[f \times id]$.

Proof: Let $\pi_1 : a \times b \to a$ be a projection in $\mathcal{C}$. Then the functor $\Sigma[\pi_1] : \mathbf{DI}[a \times b] \to \mathbf{DI}[a]$ is given by the assignment

$$\begin{array}{ccc} (S, \phi) & & \Sigma[\pi_1](S,\phi) \stackrel{df}{=} (b \times S, \mathbf{I}[\alpha_S]\phi) \\ {\scriptstyle f}\downarrow & \mapsto & \downarrow{\scriptstyle \Sigma[\pi_1]f \stackrel{df}{=} \langle \pi_2 \circ \pi_1, f\rangle \circ \alpha_S} \\ (T, \psi) & & \Sigma[\pi_1](T,\psi) \stackrel{df}{=} (b \times T, \mathbf{I}[\alpha_T]\psi) \end{array}$$

where S, T in $\mathcal{C}$, ϕ in $\mathbf{I}[(a \times b) \times S]$, ψ in $\mathbf{I}[(a \times b) \times T]$, $f : (a \times b) \times S \to T$ such that

$$\phi \sqsubseteq \mathbf{I}[\langle \pi_1, f\rangle]\psi \qquad (*)$$

and where $\alpha_X : a \times (b \times X) \to (a \times b) \times X$ is the canonical rebracketing isomorphism $\alpha_X = \langle \langle \pi_1, \pi_1 \circ \pi_2\rangle, \pi_2 \circ \pi_2\rangle$. It is easy to check that $\Sigma[\pi_1](S,\phi)$ and $\Sigma[\pi_1](T,\psi)$ both are objects in $\mathbf{DI}[a]$, and that $\Sigma[\pi_1]f$ is a morphism from $a \times (b \times S)$ to $b \times T$ in $\mathcal{C}$. So, for $\Sigma[\pi_1]f$ to be a morphism from $\Sigma[\pi_1](S,\phi)$ to $\Sigma[\pi_1](T,\psi)$ it remains to be seen that $\mathbf{I}[\alpha_S]\phi \sqsubseteq \mathbf{I}[\langle \pi_1, \Sigma[\pi_1]f\rangle]\mathbf{I}[\alpha_T]\psi$. This can be verified as follows:

$$\begin{aligned} \mathbf{I}[\alpha_S]\phi &\sqsubseteq \mathbf{I}[\alpha_S]\mathbf{I}[\langle \pi_1, f\rangle]\psi = \mathbf{I}[\langle \pi_1, f\rangle \circ \alpha_S]\psi \\ &= \mathbf{I}[\langle \pi_1 \circ \alpha_S, f \circ \alpha_S\rangle]\psi = \mathbf{I}[\langle \langle \pi_1, \pi_2 \circ \pi_1 \circ \alpha_S\rangle, f \circ \alpha_S\rangle]\psi \\ &= \mathbf{I}[\alpha_T \circ \langle \pi_1, \langle \pi_2 \circ \pi_1 \circ \alpha_S, f \circ \alpha_S\rangle\rangle]\psi \\ &= \mathbf{I}[\langle \pi_1, \langle \pi_2 \circ \pi_1 \circ \alpha_S, f \circ \alpha_S\rangle\rangle]\mathbf{I}[\alpha_T]\psi \\ &= \mathbf{I}[\langle \pi_1, \langle \pi_2 \circ \pi_1, f\rangle \circ \alpha_S\rangle]\mathbf{I}[\alpha_T]\psi = \mathbf{I}[\langle \pi_1, \Sigma[\pi_1]f\rangle]\mathbf{I}[\alpha_T]\psi. \end{aligned}$$

Thus, the mapping $\Sigma[\pi_1]$ is well-defined, and it is not difficult to show that it is in fact a functor.

In order to see that $\Sigma[\pi_1]$ is left adjoint to $\mathbf{DI}[\pi_1]$ we have to prove that there is a hom-bijection

$$[(S,\phi) \to \mathbf{DI}[\pi_1](T,\psi)]_{a\times b} \cong [\Sigma[\pi_1](S,\phi) \to (T,\psi)]_a$$

natural in both (S,ϕ) and (T,ψ), where the left homset is in $\mathbf{DI}[a \times b]$ and the right in $\mathbf{DI}[a]$. The homset equation is equivalent to

$$[(S,\phi) \to (T, \mathbf{I}[\pi_1 \times id]\psi)]_{a\times b} \cong [(b \times S, \mathbf{I}[\alpha_s]\phi) \to (T,\psi)]_a$$

by definition of $\mathbf{DI}$ and $\Sigma[\pi_1]$. We claim that such a natural hom-bijection is given by the mutually inverse mappings

$$f \mapsto f \circ \alpha_S \text{ and } g \mapsto g \circ \alpha_S^{-1}.$$

Suppose $f \in [(S,\phi) \to (T, \mathbf{I}[\pi_1 \times id]\psi)]_{a\times b}$, *i.e.* $f : (a \times b) \times S \to T$ morphism in $\mathcal{C}$ such that

$$\begin{aligned} \phi &\sqsubseteq \mathbf{I}[\langle \pi_1, f\rangle]\mathbf{I}[\pi_1 \times id]\psi = \mathbf{I}[(\pi_1 \times id) \circ \langle \pi_1, f\rangle]\psi \\ &= \mathbf{I}[\langle \pi_1 \circ \pi_1, f\rangle]\psi. \end{aligned}$$

We wish to show that the image of f, $f \circ \alpha_S$ is a morphism in the homset $[(b \times S, \mathbf{I}[\alpha_s]\phi) \to (T,\phi)]_a$. Clearly, $f \circ \alpha_S$ is a morphism in $\mathcal{C}$ from $a \times (b \times S)$ to T. We must show that $\mathbf{I}[\alpha_S]\phi \sqsubseteq \mathbf{I}[\langle \pi_1, f \circ \alpha_S\rangle]\psi$, which can be seen as follows:

$$\begin{aligned} \mathbf{I}[\alpha_S]\phi &\sqsubseteq \mathbf{I}[\alpha_S]\mathbf{I}[\langle \pi_1 \circ \pi_1, f\rangle]\psi = \mathbf{I}[\langle \pi_1 \circ \pi_1, f\rangle \circ \alpha_S]\psi \\ &= \mathbf{I}[\langle \pi_1, f \circ \alpha_S\rangle]\psi. \end{aligned}$$

Thus the first direction $f \mapsto f \circ \alpha_S$ of the bijection is well-defined. Now we show that the other direction $g \mapsto g \circ \alpha_S^{-1}$ is well-defined. Let g be in the homset $[(b \times S, \mathbf{I}[\alpha_s]\phi) \to (T,\psi)]_a$, *i.e.* $g : a \times (b \times S) \to T$ morphism in $\mathcal{C}$ such that $\mathbf{I}[\alpha_S]\phi \sqsubseteq \mathbf{I}[\langle \pi_1, g\rangle]\psi$. We must verify that the morphism $g \circ \alpha_S^{-1}$ from $(a \times b) \times S$

to T satisfies $\phi \sqsubseteq \mathbf{I}[\langle \pi_1, g \circ \alpha_S^{-1} \rangle]\mathbf{I}[\pi_1 \times id]\psi$:

$$\begin{aligned}
\phi &= \mathbf{I}[id]\phi = \mathbf{I}[\alpha_S \circ \alpha_S^{-1}]\phi = \mathbf{I}[\alpha_S^{-1}]\mathbf{I}[\alpha_S]\phi \\
&\sqsubseteq \mathbf{I}[\alpha_S^{-1}]\mathbf{I}[\langle \pi_1, g \rangle]\psi = \mathbf{I}[\langle \pi_1, g \rangle \circ \alpha_S^{-1}]\psi = \mathbf{I}[\langle \pi_1 \circ \alpha_S^{-1}, g \circ \alpha_S^{-1} \rangle]\psi \\
&= \mathbf{I}[\langle \pi_1 \circ \pi_1, g \circ \alpha_S^{-1} \rangle]\psi = \mathbf{I}[\langle \pi_1, g \circ \alpha_S^{-1} \rangle]\mathbf{I}[\pi_1 \times id]\psi.
\end{aligned}$$

It is easy to check that the hom-bijection is natural in both (S, ϕ) and (T, ψ). Naturality in (S, ϕ) comes down to the equations

$$\begin{aligned}
f \circ \alpha_S \circ \langle \pi_1, \Sigma[\pi_1]t \rangle &= f \circ \langle \pi_1, t \rangle \circ \alpha_S \\
g \circ \alpha_S^{-1} \circ \langle \pi_1, t \rangle &= g \circ \langle \pi_1, \Sigma[\pi_1]t \rangle \circ \alpha_S^{-1}
\end{aligned}$$

where $t : (a \times b) \times S' \to S$ arbitrary. Naturality in (T, ψ) follows from this and bijectivity. Thus we have shown that $\Sigma[\pi_1]$ is left adjoint to $\mathbf{DI}[\pi_1]$.

Now we verify that $\Sigma[\pi_1]$ satisfies Beck-Chevalley. Let $f : a \to a'$ be a morphism in $\mathcal{C}$, and (S, ϕ) object in $\mathbf{DI}[a \times b]$. We have $\alpha_S \circ (f \times id) = ((f \times id) \times id) \circ \alpha_S$, whence

$$\begin{aligned}
\mathbf{DI}[f]\Sigma[\pi_1](S, \phi) &= \mathbf{DI}[f](b \times S, \mathbf{I}[\alpha_S]\phi) \\
&= (b \times S, \mathbf{I}[f \times id]\mathbf{I}[\alpha_S]\phi) \\
&= (b \times S, \mathbf{I}[\alpha_S \circ (f \times id)]\phi) \\
&= (b \times S, \mathbf{I}[((f \times id) \times id) \circ \alpha_S]\phi) \\
&= (b \times S, \mathbf{I}[\alpha_S]\mathbf{I}[(f \times id) \times id]\phi) \\
&= \Sigma[\pi_1](S, \mathbf{I}[(f \times id) \times id]\phi) \\
&= \Sigma[\pi_1]\mathbf{DI}[f \times id](S, \phi)
\end{aligned}$$

The proof that

$$\mathbf{DI}[f]\Sigma[\pi_1]t = \Sigma[\pi_1]\mathbf{DI}[f \times id]t$$

for all morphisms $t \in [(S, \phi) \to (T, \psi)]_{a \times b}$ is omitted. ∎

Since we wish to interpret the (syntactic) first-order calculus of $\mathcal{L}$ in the indexed category $\mathcal{DB}_=$ we need $\mathcal{DB}_=$ to be a hyperdoctrine (at least). Of course,

since the object language $\mathcal{T}_=$ and the predicate logic $\mathcal{B}_=$ over it are folded together in $\mathcal{DB}_=$, all structure in $\mathcal{DB}_=$ will have to come from both $\mathcal{T}_=$ and $\mathcal{B}_=$. It is known that, given an indexed category $(\mathcal{C}, \mathbf{I} : \mathcal{C}^{op} \to \mathbf{Cat})$, the induced Grothendieck category has limits provided $\mathcal{C}$ and the fibres $\mathbf{C}[a]$ for all a in $\mathcal{C}$ have limits, and provided limits in the fibres are preserved by translation functors [TBG89]. But what about the more general construction of $\mathcal{DB}$? In general, we find

Theorem 5.2.13 *Let $\mathcal{I} = (\mathcal{C}, \mathbf{I} : \mathcal{C}^{op} \to \mathbf{PreOrd})$ be a hyperdoctrine where all fibres are preodered sets, and let $\mathcal{C}$ be bicartesian closed with products $\times$ and sums $+$. Further, assume that translations $\mathbf{I}[id \times \iota_1]$ along the canonical morphisms $id \times \iota_1 : a \times b \to a \times (b + c)$ have left adjoints $\exists[id \times \iota_1] : \mathbf{I}[a \times b] \to \mathbf{I}[a \times (b + c)]$ such that Beck-Chevalley is satisfied for pull-back squares*

$$\begin{array}{ccc} a \times b & \xrightarrow{id \times \iota_1} & a \times (b + c) \\ {\scriptstyle f \times id}\downarrow & & \downarrow{\scriptstyle f \times id} \\ a' \times b & \xrightarrow[id \times \iota_1]{} & a' \times (b + c) \end{array}$$

where $f : a \to a'$ arbitrary, i.e. we have

$$\exists[id \times \iota_1]\mathbf{I}[f \times id] \cong \mathbf{I}[f \times id]\exists[id \times \iota_1]$$

Then, the indexed category $\mathcal{DI} = (\mathcal{C}, \mathbf{DI} : \mathcal{C}^{op} \to \mathbf{Cat})$ as constructed in Definition 5.2.10 is a hyperdoctrine.

Proof: A word on notation: as in Definition 5.2.10 the objects in the base category $\mathcal{C}$ of $\mathcal{I}$ and $\mathcal{DI}$ will be denoted by lower case letters a, b, c, *etc.* the objects in the fibres $\mathbf{I}[a]$ by lower case Greek letters ϕ, ψ, *etc.* and finally the objects in the fibres $\mathbf{DI}[a]$ will be referred to as pairs $(S, \phi), (T, \psi)$ where the first components, denoted by upper case letters S, T, *etc.* are objects in $\mathcal{C}$ and ϕ, ψ, *etc.* are objects in $\mathbf{I}[a \times S]$. To refer to the bicartesian structure of $\mathcal{C}$ we use the symbols $\mathbf{1}, \times, \Rightarrow, \mathbf{0}, +$ and for the bicartesian structure of the fibres $\mathbf{I}[b]$ the symbols *true*, $\wedge, \supset$, *false*, $\vee$.

First, given a in $\mathcal{C}$ it is shown that $\mathbf{DI}[a]$ is bicartesian closed. We need to identify in $\mathbf{DI}[a]$ a terminal object *true*, products $(S,\phi) \wedge (T,\psi)$, exponentials $(S,\phi) \supset (T,\psi)$, initial object *false*, and sums $(S,\phi) \vee (T,\psi)$. These are obtained from the bicartesian structure of the base category $\mathcal{C}$ and of the fibres $\mathbf{I}[b]$ in the way described below:

TERMINAL OBJECT. The terminal object in $\mathbf{DI}[a]$ is the pair $(\mathbf{1}, \mathit{true})$ where $\mathbf{1}$ is the terminal object in $\mathcal{C}$ and *true* is terminal in $\mathbf{I}[a \times \mathbf{1}]$. For (S,ϕ) in $\mathbf{DI}[a]$ the unique morphism $!_{(S,\phi)}$ into $(\mathbf{1}, \mathit{true})$ is the unique morphism $!_{a \times S}$ in $\mathcal{C}$.

PRODUCTS. The product of objects (S,ϕ) and (T,ψ) in $\mathbf{DI}[a]$ is the pair

$$(S,\phi) \wedge (T,\psi) \stackrel{df}{=} (S \times T\,,\, \mathbf{I}[id_a \times \pi_1]\phi \wedge \mathbf{I}[id_a \times \pi_2]\psi)$$

where $\times$ is product in $\mathcal{C}$ and $\wedge$ product in $\mathbf{I}[a \times (S \times T)]$. The projections are

$$\begin{aligned} \pi_1 \circ \pi_2 &: (S,\phi) \wedge (T,\psi) \rightarrow (S,\phi) \\ \pi_2 \circ \pi_2 &: (S,\phi) \wedge (T,\psi) \rightarrow (T,\psi). \end{aligned}$$

Given morphisms $t_1 : (S,\phi) \rightarrow (T_1,\psi_1)$ and $t_2 : (S,\phi) \rightarrow (T_2,\psi_2)$ in $\mathbf{DI}[a]$ their pairing is given by the morphism

$$\langle t_1, t_2 \rangle : (S,\phi) \rightarrow ((T_1,\psi_1) \wedge (T_2,\psi_2))$$

where $\langle , \rangle$ is the pairing in $\mathcal{C}$. The simple proof that these data satisfy the standard product equations is omitted.

EXPONENTIALS. The exponent of objects (S,ϕ) and (T,ψ) in $\mathbf{DI}[a]$ is the pair

$$(S,\phi) \supset (T,\psi) \stackrel{df}{=} (S \Rightarrow T\,,\, \forall[\pi_1](\mathbf{I}[\mathit{proj}]\phi \supset \mathbf{I}[(id_a \times \mathit{eval}) \circ \mathit{swap}]\psi))$$

where $\Rightarrow$ is exponential in $\mathcal{C}$, $\mathit{eval} : S \times (S \Rightarrow T) \rightarrow T$ its associated evaluation, $\supset$ exponential in $\mathbf{I}[(a \times (S \Rightarrow T)) \times S]$, and $\forall[\pi_1]$ is right adjoint to $\mathbf{I}[\pi_1]$. *proj* and *swap* are auxiliary morphisms abbreviating

$$\begin{aligned} \mathit{proj} &\stackrel{df}{=} \langle \pi_1 \circ \pi_1, \pi_2 \rangle : (a \times (S \Rightarrow T)) \times S \rightarrow a \times S \\ \mathit{swap} &\stackrel{df}{=} \langle \pi_1 \circ \pi_1, \langle \pi_2, \pi_2 \circ \pi_1 \rangle \rangle : (a \times (S \Rightarrow T)) \times S \rightarrow a \times (S \times (S \Rightarrow T)). \end{aligned}$$

The evaluation map $eval : (S,\phi) \wedge ((S,\phi) \supset (T,\psi)) \to (T,\psi)$ is the morphism $eval \circ \pi_2 : a \times (S \times (S \Rightarrow T)) \to T$. Finally, given a morphism $t : (S,\phi) \wedge (T,\psi) \to (U,\theta)$ in $\mathbf{DI}[a]$ we need to define its 'currying' $curry(t) : (S,\phi) \to (T,\psi) \supset (U,\theta)$. This morphism $curry(t)$ is given by

$$curry(t \circ \alpha^{-1}) : a \times S \to (T \Rightarrow U)$$

where *curry* is currying in $\mathcal{C}$, and $\alpha^{-1} : (a \times S) \times T \to a \times (S \times T)$ the obvious rebracketing. The simple proof that these data satisfy the standard equations for function spaces is omitted.

INITIAL OBJECT. The initial object in $\mathbf{DI}[a]$ is the pair $(0, false)$ where 0 is initial in $\mathcal{C}$ and *false* the initial object in $\mathbf{I}[a \times 0]$. For (S,ϕ) in $\mathbf{DI}[a]$ the unique morphism $\Box_{(S,\phi)}$ from $(0, false)$ into (S,ϕ) is the morphism $\Box_S \circ \pi_2$ in $\mathcal{C}$, where $\Box_S : 0 \to S$ is the unique initial morphism into S.

SUMS. The sum of objects (S,ϕ) and (T,ψ) in $\mathbf{DI}[a]$ is the pair

$$(S,\phi) \vee (T,\psi) \stackrel{df}{=} (S+T,\ \exists[id_a \times \iota_1]\phi \vee \mathbf{I}[id_a \times flop]\exists[id_a \times \iota_1]\psi)$$

where ι_1 are the injections $S \to S+T$ and $T \to T+S$ as appropriate, and $flop : S+T \to T+S$ is the canonical isomorphism $[\iota_2, \iota_1]$, $\vee$ is sum in $\mathbf{I}[a \times (S+T)]$, and $\exists[id_a \times \iota_1]$ left adjoint to $\mathbf{I}[id_a \times \iota_1]$. The injections are

$$\begin{aligned} \iota_1 \circ \pi_2 &: (S,\phi) \to (S,\phi) \vee (T,\psi) \\ \iota_2 \circ \pi_2 &: (T,\psi) \to (S,\phi) \vee (T,\psi) \end{aligned}$$

which, as morphisms in $\mathbf{DI}[a]$, will be denoted by ι_1, ι_2. The sum of two morphisms $s : (S,\phi) \to (U,\theta)$ and $t : (T,\psi) \to (U,\theta)$ is the morphism

$$[s,t] \circ \sigma : (S,\phi) \vee (T,\psi) \to (U,\theta)$$

where $\sigma : (a \times (S+T)) \to ((a \times S) + (a \times T))$ is the canonical distribution map that exists in any bicartesian closed category; it can be defined as

$$\sigma \stackrel{df}{=} eval \circ \langle \pi_1, [curry(\iota_1 \circ flip), curry(\iota_2 \circ flip)] \circ \pi_2 \rangle$$

with *flip* abbreviating the transposition map $\langle \pi_2, \pi_1 \rangle$. The sum of s and t in $\mathbf{DI}[a]$, will be written $[s, t]$.

Before we come to verify that these definitions accomplish what they should we note that $\sigma^{-1} = [id \times \iota_1, id \times \iota_2]$ is the inverse of σ. It will be used several times below. To show that it is right inverse we compute

$$\begin{aligned}
\sigma \circ (id \times \iota_1) &= eval \circ \langle \pi_1, [curry(\iota_1 \circ flip), curry(\iota_2 \circ flip)] \circ \pi_2 \rangle \circ (id \times \iota_1) \\
&= eval \circ \langle \pi_1, [curry(\iota_1 \circ flip), curry(\iota_2 \circ flip)] \circ \iota_1 \circ \pi_2 \rangle \\
&= eval \circ \langle \pi_1, curry(\iota_1 \circ flip) \circ \pi_2 \rangle \\
&= eval \circ \langle \pi_2, curry(\iota_1 \circ flip) \circ \pi_1 \rangle \circ flip \\
&= \iota_1 \circ flip \circ flip \\
&= \iota_1
\end{aligned}$$

and similarly, $\sigma \circ (id \times \iota_2) = \iota_2$, whence $\sigma \circ \sigma^{-1} = \sigma \circ [id \times \iota_1, id \times \iota_2] = [\sigma \circ (id \times \iota_1), \sigma \circ (id \times \iota_2)] = [\iota_1, \iota_2] = id$. For the other direction let the morphism $[curry(\iota_1 \circ flip), curry(\iota_2 \circ flip)]$ be abbreviated by π, so that $\sigma = eval \circ (id \times \pi)$. We observe that

$$\begin{aligned}
& curry(\sigma^{-1} \circ \sigma \circ flip) \circ \iota_1 \\
&= curry(\sigma^{-1} \circ eval \circ (id \times \pi) \circ flip) \circ \iota_1 \\
&= curry(\sigma^{-1} \circ eval \circ (id \times \pi) \circ flip \circ (\iota_1 \times id)) && (5.2) \\
&= curry(\sigma^{-1} \circ eval \circ (\pi_2, \pi \circ \iota_1 \circ \pi_1)) \\
&= curry(\sigma^{-1} \circ eval \circ \langle \pi_2, curry(\iota_1 \circ flip) \circ \pi_1 \rangle) \\
&= curry(\sigma^{-1} \circ \iota_1 \circ flip) \\
&= curry((id \times \iota_1) \circ flip) \\
&= curry(flip \circ (\iota_1 \times id)) \\
&= curry(flip) \circ \iota_1. && (5.3)
\end{aligned}$$

Lines (5.2) and (5.3) exploit the fact that in cartesian closed categories currying is 'natural', *i.e.* for all morphisms $f : X \times Y \to Z$ and $g : X' \to X$ we have

$curry(f)\circ g = curry(f\circ(g\times id_Y))$. Similarly we find $curry(\sigma^{-1}\circ\sigma\circ flip)\circ\iota_2 = curry(flip)\circ\iota_2$. Hence,

$$\begin{aligned}
&\sigma^{-1}\circ\sigma\\
&= \sigma^{-1}\circ\sigma\circ flip\circ flip\\
&= uncurry(curry(\sigma^{-1}\circ\sigma\circ flip))\circ flip && (5.4)\\
&= uncurry([curry(\sigma^{-1}\circ\sigma\circ flip)\circ\iota_1, curry(\sigma^{-1}\circ\sigma\circ flip)\circ\iota_2])\circ flip\\
&= uncurry([curry(flip)\circ\iota_1, curry(flip)\circ\iota_2])\circ flip\\
&= uncurry(curry(flip))\circ flip\\
&= id && (5.5)
\end{aligned}$$

where for any $f : X\to(Z^Y)$, $uncurry(f) : X\times Y\to Z$ is defined by $uncurry(f) \stackrel{df}{=} eval\circ\langle\pi_2, f\circ\pi_1\rangle$. It holds that for all $g : X\times Y\to Z$, $uncurry(curry(g)) = g$ by the laws for function spaces. This justifies lines (5.4) and (5.5) above. All the other lines are straightforward in applying the fundamental properties of products and sums. Thus, we find that σ and σ^{-1} are mutual inverse.

Now, let us check that the categorical sum in $\mathbf{DI}[a]$ is well-defined. For $\iota_1\circ\pi_2 : a\times S\to(S+T)$ to be a well-defined morphism from (S,ϕ) into $(S,\phi)\vee(T,\psi)$ in fibre $\mathbf{DI}[a]$ it must satisfy

$$\phi \sqsubseteq \mathbf{I}[\langle\pi_1,\iota_1\circ\pi_2\rangle](\exists[id\times\iota_1]\phi\vee\mathbf{I}[id\times flop]\exists[id\times\iota_1]\psi)$$

which, since $\mathbf{I}[\langle\pi_1,\iota_1\circ\pi_2\rangle] = \mathbf{I}[id\times\iota_1]$ and $\exists[id\times\iota_1]$ left adjoint to $\mathbf{I}[id\times\iota_1]$ is equivalent to

$$\exists[id\times\iota_1]\phi \sqsubseteq \exists[id\times\iota_1]\phi\vee\mathbf{I}[id\times flop]\exists[id\times\iota_1]\psi.$$

But this holds because $\vee$ is sum in $\mathbf{I}[a\times(S+T)]$. Analogously, it can be seen that $\iota_2\circ\pi_2 : a\times T\to(S+T)$ is a well-defined morphism from (T,ψ) into $(S,\phi)\vee(T,\psi)$ in $\mathbf{DI}[a]$, *i.e.*

$$\psi \sqsubseteq \mathbf{I}[\langle\pi_1,\iota_2\circ\pi_2\rangle](\exists[id\times\iota_1]\phi\vee\mathbf{I}[id\times flop]\exists[id\times\iota_1]\psi).$$

For this is equivalent to

$$\psi \sqsubseteq \mathbf{I}[id \times \iota_2]\exists[id \times \iota_1]\phi \vee \mathbf{I}[id \times \iota_2]\mathbf{I}[id \times flop]\exists[id \times \iota_1]\psi$$

since $\mathbf{I}[\langle \pi_1, \iota_2 \circ \pi_2 \rangle] = \mathbf{I}[id \times \iota_2]$ and since translations, in this case $\mathbf{I}[id \times \iota_2]$, preserve bicartesian structure. Now, $\mathbf{I}[id \times \iota_2]\mathbf{I}[id \times flop] = \mathbf{I}[(id \times flop) \circ (id \times \iota_2)] = \mathbf{I}[id \times \iota_1]$, so by the properties of $\vee$ is suffices to show

$$\psi \sqsubseteq \mathbf{I}[id \times \iota_1]\exists[id \times \iota_1]\psi$$

which is immediate as $\exists[id \times \iota_1]$ left adjoint to $\mathbf{I}[id \times \iota_1]$. Finally, assume that $s : (S, \phi) \to (U, \theta)$ and $t : (T, \psi) \to (U, \theta)$ are well-defined morphisms in $\mathbf{DI}[a]$, *i.e.* $s : a \times S \to U$, $t : a \times T \to T$, and

$$\phi \sqsubseteq \mathbf{I}[\langle \pi_1, s \rangle]\theta \qquad \psi \sqsubseteq \mathbf{I}[\langle \pi_1, t \rangle]\theta.$$

We must check that the sum $[s, t] \circ \sigma : a \times (S + T) \to U$ of s and t is a well-defined morphism from $(S, \phi) \vee (T, \psi)$ to (U, θ). This comes down to the condition

$$\exists[id \times \iota_1]\phi \vee \mathbf{I}[id \times flop]\exists[id \times \iota_1]\psi \sqsubseteq \mathbf{I}[\langle \pi_1, [s, t] \circ \sigma \rangle]\theta$$

which by the properties of $\vee$ is equivalent to the two inequations

$$\exists[id \times \iota_1]\phi \sqsubseteq \mathbf{I}[\langle \pi_1, [s, t] \circ \sigma \rangle]\theta \tag{5.6}$$

$$\mathbf{I}[id \times flop]\exists[id \times \iota_1]\psi \sqsubseteq \mathbf{I}[\langle \pi_1, [s, t] \circ \sigma \rangle]\theta. \tag{5.7}$$

Now, the first of these is equivalent to

$$\phi \sqsubseteq \mathbf{I}[\langle \pi_1, [s, t] \circ \sigma \rangle \circ (id \times \iota_1)]\theta$$

since $\exists[id \times \iota_1]$ left adjoint to $\mathbf{I}[id \times \iota_1]$ and functoriality of $\mathbf{I}[id \times \iota_1]$. But this is precisely the assumption $\phi \sqsubseteq \mathbf{I}[\langle \pi_1, s \rangle]\theta$, for $\langle \pi_1, [s, t] \circ \sigma \rangle \circ (id \times \iota_1) = \langle \pi_1, [s, t] \circ \sigma \circ (id \times \iota_1) \rangle = \langle \pi_1, [s, t] \circ \iota_1 \rangle = \langle \pi_1, s \rangle$. Hence we have shown (5.6). Applying $\mathbf{I}[id \times flop]$ to both sides turns (5.7) into the equivalent

$$\exists[id \times \iota_1]\psi \sqsubseteq \mathbf{I}[\langle \pi_1, [s, t] \circ \sigma \circ (id \times flop) \rangle]\theta \tag{5.8}$$

noting that $\mathbf{I}[id \times flop]\mathbf{I}[id \times flop] = \mathbf{I}[(id \times flop) \circ (id \times flop)] = \mathbf{I}[id]$ is the identity map, and $\mathbf{I}[id \times flop]\mathbf{I}[\langle \pi_1, [s,t] \circ \sigma \rangle] = \mathbf{I}[\langle \pi_1, [s,t] \circ \sigma \rangle \circ (id \times flop)] = \mathbf{I}[\langle \pi_1, [s,t] \circ \sigma \circ (id \times flop) \rangle]$. The right side of (5.8) can be simplified, using the identity

$$\begin{aligned}
[s,t] \circ \sigma \circ (id \times flop) &= [s,t] \circ ((id \times flop) \circ \sigma^{-1})^{-1} \\
&= [s,t] \circ ((id \times flop) \circ [id \times \iota_1, id \times \iota_2])^{-1} \\
&= [s,t] \circ ([(id \times flop) \circ (id \times \iota_1), (id \times flop) \circ (id \times \iota_2)])^{-1} \\
&= [s,t] \circ ([id \times \iota_2, id \times \iota_1])^{-1} \\
&= [s,t] \circ ([id \times \iota_1, id \times \iota_2] \circ flop)^{-1} \\
&= [s,t] \circ (\sigma^{-1} \circ flop)^{-1} \\
&= [s,t] \circ flop \circ \sigma \\
&= [t,s] \circ \sigma
\end{aligned}$$

to become

$$\exists[id \times \iota_1]\psi \sqsubseteq \mathbf{I}[\langle \pi_1, [t,s] \circ \sigma \rangle]\theta.$$

But this follows from the assumption $\psi \sqsubseteq \mathbf{I}[\langle \pi_1, t \rangle]\theta$ since $\exists[id \times \iota_1]$ left adjoint to $\mathbf{I}[id \times \iota_1]$ and $\mathbf{I}[id \times \iota_1]\mathbf{I}[\langle \pi_1, [t,s] \circ \sigma \rangle] = \mathbf{I}[\langle \pi_1, [t,s] \circ \sigma \rangle \circ (id \times \iota_1)] = \mathbf{I}[\langle \pi_1, t \rangle]$. Thus we have shown (5.7) whence the categorical sum in $\mathbf{DI}[a]$ is well-defined.

Now we wish to verify that these data indeed satisfy the universal properties of a categorical sum in $\mathbf{DI}[a]$. The three equations that we must prove are

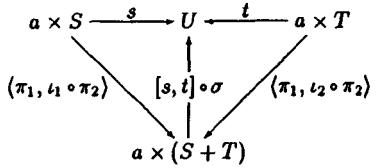
$$[s,t] \circ \iota_1 = s \tag{5.9}$$

$$[s,t] \circ \iota_2 = t \tag{5.10}$$

$$[t \circ \iota_1, t \circ \iota_2] = t \tag{5.11}$$

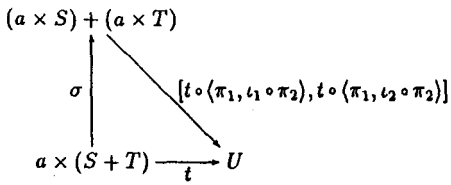
which constitute the characterizing β- and η-laws for categorical sums. Note, these equations are to be read as equations in $\mathbf{DI}[a]$, *i.e.* $\iota_1, \iota_2, [\cdot,\cdot]$ are the injections and sum defined above, and composition $\circ$ is composition of morphisms in $\mathbf{DI}[a]$.

Ad (5.9), (5.10). By definition, (5.9) and (5.10) amount to proving that the diagram



commutes for all s, t. To this end we first recall that $\sigma \circ (id \times \iota_1) = \iota_1$ and $\sigma \circ (id \times \iota_2) = \iota_2$. Hence, we get $[s,t] \circ \sigma \circ \langle \pi_1, \iota_1 \circ \pi_2 \rangle = [s,t] \circ \sigma \circ (id \times \iota_1) = [s,t] \circ \iota_1 = s$ which precisely says that the left side of the diagram commutes. The other side of the diagram is dealt with similarly.

Ad (5.11). By definition, (5.11) amounts to proving that the diagram



commutes in $\mathcal{C}$ for all t. Let the morphism $[t \circ \langle \pi_1, \iota_1 \circ \pi_2 \rangle, t \circ \langle \pi_1, \iota_2 \circ \pi_2 \rangle]$ in the diagram be abbreviated by π. In the following we will show the currified version of the diagram, *i.e.* the equation

$$curry(\pi \circ \sigma \circ flip) = curry(t \circ flip). \tag{5.12}$$

From this, then, we immediately get the original diagram since

$$\begin{aligned} \pi \circ \sigma &= \pi \circ \sigma \circ flip \circ flip \\ &= uncurry(curry(\pi \circ \sigma \circ flip)) \circ flip \\ &= uncurry(curry(t \circ flip)) \circ flip \\ &= t \circ flip \circ flip \\ &= t. \end{aligned}$$

Now, let us prove (5.12):

$$
\begin{aligned}
& curry(\pi \circ \sigma \circ flip) \\
&= [curry(\pi \circ \sigma \circ flip) \circ \iota_1, curry(\pi \circ \sigma \circ flip) \circ \iota_2] \\
&= [curry(\pi \circ \sigma \circ flip \circ (\iota_1 \times id)), curry(\pi \circ \sigma \circ flip \circ (\iota_2 \times id))] \\
&= [curry(\pi \circ \sigma \circ (id \times \iota_1) \circ flip), curry(\pi \circ \sigma \circ (id \times \iota_2) \circ flip)] \\
&= [curry(\pi \circ \iota_1 \circ flip), curry(\pi \circ \iota_2 \circ flip)] \\
&= [curry(t \circ \langle \pi_1, \iota_1 \circ \pi_2 \rangle \circ flip), curry(t \circ \langle \pi_1, \iota_2 \circ \pi_2 \rangle \circ flip)] \\
&= [curry(t \circ flip \circ (\iota_1 \times id)), curry(t \circ flip \circ (\iota_2 \times id))] \\
&= [curry(t \circ flip) \circ \iota_1, curry(t \circ flip) \circ \iota_2] \\
&= curry(t \circ flip).
\end{aligned}
$$

Thus, we have shown that the fibres $\mathbf{DI}[a]$ are bicartesian closed. The proof that this structure is preserved by translation functors is omitted. We remark that the assumption in the statement of the theorem that $\exists[id \times \iota_1]$ satisfy Beck-Chevalley is used to show that sums are preserved by translations. Next, it is verified that there exist left and right adjoints of translations along first projections:

EXISTENTIALS. Let $\pi_1 : a \times b \to a$ be a first projection in $\mathcal{C}$. Then, by Lemma 5.2.12 the translation functor $\mathbf{I}[\pi_1] : \mathbf{I}[a] \to \mathbf{I}[a \times b]$ has a left adjoint $\Sigma[\pi_1]$ satisfying the strong Beck-Chevalley Condition for pull-back squares induced by first projections.

UNIVERSALS. Let $\pi_1 : a \times b \to a$ be a first projection in $\mathcal{C}$. Then, the assignment

$$
\begin{array}{ccc}
(S, \phi) & & (b \Rightarrow S, \forall[\pi_1]\mathbf{I}[app]\phi) \\
\Big\downarrow t & \mapsto & \Big\downarrow curry(t \circ app) \\
(T, \psi) & & (b \Rightarrow T, \forall[\pi_1]\mathbf{I}[app]\psi)
\end{array}
$$

where $\pi_1 : (a \times (b \Rightarrow X)) \times b \to a \times (b \Rightarrow X)$, $X = S$ or $X = T$ as appropriate, are first projections and $app : (a \times (b \Rightarrow X)) \times b \to (a \times b) \times X$ stands for

$$app \stackrel{df}{=} \langle (\pi_1 \circ \pi_1, \pi_2), eval \circ \langle \pi_2, \pi_2 \circ \pi_1 \rangle \rangle$$

defines a right adjoint to translation functor $\mathbf{DI}[\pi_1] : \mathbf{DI}[a] \to \mathbf{DI}[a \times b]$. It will be denoted by $\Pi[\pi_1]$.

First, we check that the assignment is well-defined. It is easy to see that if (S, ϕ), (T, ψ) objects in $\mathbf{DI}[a \times b]$, then $\Pi[\pi_1](S, \phi) = (b \Rightarrow S, \forall[\pi_1]\mathbf{I}[app]\phi)$ and $\Pi[\pi_1](T, \psi) = (b \Rightarrow T, \forall[\pi_1]\mathbf{I}[app]\psi)$ as defined are indeed objects in $\mathbf{DI}[a]$. Also, it is obvious that if $t : (a \times b) \times S \to T$, then $\Pi[\pi_1](t) = curry(t \circ app)$ is a morphism in $\mathcal{C}$ from $a \times (b \Rightarrow S)$ to $b \Rightarrow T$. For $\Pi[\pi_1](t)$ to be a well-defined morphism

$$curry(t \circ app) : (b \Rightarrow S, \forall[\pi_1]\mathbf{I}[app]\phi) \to (b \Rightarrow T, \forall[\pi_1]\mathbf{I}[app]\psi)$$

it remains to be seen that it satisfies

$$\forall[\pi_1]\mathbf{I}[app]\phi \;\subseteq\; \mathbf{I}[\langle \pi_1, curry(t \circ app) \rangle]\forall[\pi_1]\mathbf{I}[app]\psi. \tag{5.13}$$

The right side of the inequation can be simplified using the fact that $\forall[\pi_1]$ satisfies the Beck-Chevalley Condition, *viz.* we observe that

$$\begin{array}{ccc} (a \times (b \Rightarrow S)) \times b & \xrightarrow{\pi_1} & a \times (b \Rightarrow S) \\ \downarrow{\scriptstyle \langle \pi_1, \pi \rangle \times id} & & \downarrow{\scriptstyle \langle \pi_1, \pi \rangle} \\ (a \times (b \Rightarrow T)) \times b & \xrightarrow[\pi_1]{} & a \times (b \Rightarrow T) \end{array}$$

is a pull-back diagram induced by a first projection, where here and in the following π is an abbreviation for $curry(t \circ app)$. Beck-Chevalley implies that $\mathbf{I}[\langle \pi_1, \pi \rangle]\forall[\pi_1] \cong \forall[\pi_1]\mathbf{I}[\langle \pi_1, \pi \rangle \times id]$, whence (5.13) is equivalent to inequation

$$\forall[\pi_1]\mathbf{I}[app]\phi \;\subseteq\; \forall[\pi_1]\mathbf{I}[\langle \pi_1, \pi \rangle \times id]\psi.$$

Now, since $\forall[\pi_1]$ is monotone this reduces to

$$\mathbf{I}[app]\phi \;\subseteq\; \mathbf{I}[\langle \pi_1, \pi \rangle \times id]\mathbf{I}[app]\psi. \tag{5.14}$$

Again, the right side can be simplified, *viz.* we compute

$$\begin{aligned}
& app \circ (\langle \pi_1, \pi \rangle \times id) \\
&= \langle \langle \pi_1 \circ \pi_1, \pi_2 \rangle, eval \circ \langle \pi_2, \pi_2 \circ \pi_1 \rangle \rangle \circ (\langle \pi_1, \pi \rangle \times id) \\
&= \langle \langle \pi_1 \circ \pi_1, \pi_2 \rangle \circ (\langle \pi_1, \pi \rangle \times id), eval \circ \langle \pi_2, \pi_2 \circ \pi_1 \rangle \circ (\langle \pi_1, \pi \rangle \times id) \rangle \\
&= \langle \langle \pi_1 \circ \pi_1, \pi_2 \rangle, eval \circ \langle \pi_2, \pi \circ \pi_1 \rangle \rangle \\
&= \langle \pi_1 \circ app, eval \circ \langle \pi_2, curry(t \circ app) \circ \pi_1 \rangle \rangle \\
&= \langle \pi_1 \circ app, t \circ app \rangle \\
&= \langle \pi_1, t \rangle \circ app
\end{aligned}$$

so that

$$\mathbf{I}[\langle \pi_1, \pi \rangle \times id]\mathbf{I}[app] = \mathbf{I}[app \circ (\langle \pi_1, \pi \rangle \times id)] = \mathbf{I}[\langle \pi_1, t \rangle \circ app] = \mathbf{I}[app]\mathbf{I}[\langle \pi_1, t \rangle].$$

This together with monotonicity of $\mathbf{I}[app]$ reduces inequation (5.14) to

$$\phi \sqsubseteq \mathbf{I}[\langle \pi_1, t \rangle]\psi$$

which immediately follows from the assumption that t is a morphism from (S, ϕ) to (T, ψ) in $\mathbf{DI}[a \times b]$. This completes the proof of (5.13).

Next we check that the map $\Pi[\pi_1] : \mathbf{DI}[a \times b] \to \mathbf{DI}[a]$ is a functor. Let $id_{(S,\phi)} = \pi_2 : (S, \phi) \to (S, \phi)$ be an identity in $\mathbf{DI}[a \times b]$ (*cf.* Definition 5.2.10), then

$$\begin{aligned}
\Pi[\pi_1]id_{(S,\phi)} &= curry(\pi_2 \circ app) = curry(eval \circ \langle \pi_2, \pi_2 \circ \pi_1 \rangle) \\
&= curry(uncurry(\pi_2)) = \pi_2 \\
&= id_{(S,\phi)}.
\end{aligned}$$

Further, let $t : (S, \phi) \to (T, \psi)$ and $s : (T, \psi) \to (U, \theta)$ be morphisms in $\mathbf{DI}[a \times b]$. We wish to show that $\Pi[\pi_1](s \circ t) = \Pi[\pi_1](s) \circ \Pi[\pi_1](t)$ holds in $\mathbf{DI}[a]$. Unraveling the definitions, this comes down to proving

$$curry(s \circ \langle \pi_1, t \rangle \circ app) = curry(s \circ app) \circ \langle \pi_1, curry(t \circ app) \rangle$$

in $\mathcal{C}$, which is done as follows, where again π abbreviates $curry(t \circ app)$:

$$\begin{aligned}
&curry(s \circ app) \circ \langle \pi_1, \pi \rangle \\
&= curry(s \circ app \circ (\langle \pi_1, \pi \rangle \times id)) \\
&= curry(s \circ \langle \langle \pi_1 \circ \pi_1, \pi_2 \rangle \circ (\langle \pi_1, \pi \rangle \times id), eval \circ \langle \pi_2, \pi_2 \circ \pi_1 \rangle \circ (\langle \pi_1, \pi \rangle \times id))) \\
&= curry(s \circ \langle \langle \pi_1 \circ \pi_1, \pi_2 \rangle, eval \circ \langle \pi_2, \pi \circ \pi_1 \rangle)) \\
&= curry(s \circ \langle \pi_1 \circ app, t \circ app \rangle) \\
&= curry(s \circ \langle \pi_1, t \rangle \circ app).
\end{aligned}$$

Now we verify that the functor $\Pi[\pi_1] : \mathbf{DI}[a \times b] \to \mathbf{DI}[a]$ is right adjoint to $\mathbf{DI}[\pi_1]$. The goal is to establish a hom-bijection

$$[(S, \phi) \to \Pi[\pi_1](T, \psi)]_a \;\cong\; [\mathbf{DI}[\pi_1](S, \phi) \to (T, \psi)]_{a \times b} \qquad (5.15)$$

where the left homset is taken in $\mathbf{DI}[a]$, the right in $\mathbf{DI}[a \times b]$, and to prove that it is natural in both (S, ϕ) and (T, ψ). After eliminating the definitions the hom-equation turns into

$$[(S, \phi) \to (b \Rightarrow T, \forall[\pi_1]\mathbf{I}[app]\psi)]_a \;\cong\; [(S, \mathbf{I}[\pi_1 \times id]\phi) \to (T, \psi)]_{a \times b}$$

We claim that such a bijection is given by the maps

$$\begin{aligned}
s &\mapsto s^\flat \stackrel{df}{=} eval \circ (id \times s) \circ swap \\
t &\mapsto t^\sharp \stackrel{df}{=} curry(t \circ swap^{-1} \circ flip)
\end{aligned}$$

where $swap : (a \times b) \times S \to b \times (a \times S)$ and its inverse $swap^{-1}$ are the canonical morphisms that permute and rebracket their arguments in the apparent way. Let us check first that the maps $s \mapsto s^\flat$ and $t \mapsto t^\sharp$ are well-defined. Suppose, s is a morphism in the homset $[(S, \phi) \to (b \Rightarrow T, \forall[\pi_1]\mathbf{I}[app]\psi)]_a$, *i.e.* $s : a \times S \to (b \Rightarrow T)$ and

$$\phi \;\sqsubseteq\; \mathbf{I}[\langle \pi_1, s \rangle]\forall[\pi_1]\mathbf{I}[app]\psi. \qquad (5.16)$$

Clearly, $s^\flat$ is a morphism from $(a \times b) \times S$ to T, for which we must show

$$\mathbf{I}[\pi_1 \times id]\phi \;\sqsubseteq\; \mathbf{I}[\langle \pi_1, s^\flat \rangle]\psi.$$

This is readily computed:

$$\begin{aligned}
&\mathbf{I}[\pi_1 \times id]\phi \\
&\sqsubseteq\ \mathbf{I}[\pi_1 \times id]\mathbf{I}[\langle \pi_1, s\rangle]\forall[\pi_1]\mathbf{I}[app]\psi && (5.17)\\
&=\ \mathbf{I}[\langle \pi_1, s\rangle \circ (\pi_1 \times id)]\forall[\pi_1]\mathbf{I}[app]\psi \\
&=\ \mathbf{I}[\pi_1 \circ (\langle \pi_1, s\rangle \times id) \circ flip \circ swap]\forall[\pi_1]\mathbf{I}[app]\psi && (5.18)\\
&=\ \mathbf{I}[(\langle \pi_1, s\rangle \times id) \circ flip \circ swap]\mathbf{I}[\pi_1]\forall[\pi_1]\mathbf{I}[app]\psi \\
&\sqsubseteq\ \mathbf{I}[(\langle \pi_1, s\rangle \times id) \circ flip \circ swap]\mathbf{I}[app]\psi && (5.19)\\
&=\ \mathbf{I}[app \circ (\langle \pi_1, s\rangle \times id) \circ flip \circ swap]\psi \\
&=\ \mathbf{I}[\langle \pi_1, s^\flat\rangle]\psi && (5.20)
\end{aligned}$$

Inequation (5.17) is a consequence of the assumption (5.16) and monotonicity of $\mathbf{I}[\pi_1 \times id]$. Equations (5.18) and (5.20) cut short a number of simple computations involving products, and inequation (5.19) follows since $\forall[\pi_1]$ is right adjoint to $\mathbf{I}[\pi_1]$. Thus, we have shown that $s \mapsto s^\flat$ is well-defined. Now to map $t \mapsto t^\natural$: Suppose given a morphism t in the homset $[(S, \mathbf{I}[\pi_1 \times id]\phi) \to (T, \psi)]_{a\times b}$, *i.e.* $t : (a \times b) \times S \to T$ and

$$\mathbf{I}[\pi_1 \times id]\phi \ \sqsubseteq\ \mathbf{I}[\langle \pi_1, t\rangle]\psi. \tag{5.21}$$

$t^\natural$ is a morphism from $a \times S \to (b \Rightarrow T)$, which is a morphism in $\mathbf{DI}[a]$ from (S, ϕ) to $(b \Rightarrow T, \forall[\pi_1]\mathbf{I}[app]\psi)$ if it holds that

$$\phi \ \sqsubseteq\ \mathbf{I}[\langle \pi_1, t^\natural\rangle]\forall[\pi_1]\mathbf{I}[app]\psi. \tag{5.22}$$

By Beck-Chevalley for $\forall[\pi_1]$ we have $\mathbf{I}[\langle \pi_1, t^\natural\rangle]\forall[\pi_1] \cong \forall[\pi_1]\mathbf{I}[\langle \pi_1, t^\natural\rangle \times id]$ since

$$\begin{array}{ccc}
(a \times S) \times b & \xrightarrow{\ \pi_1\ } & a \times S \\
{\scriptstyle \langle \pi_1, t^\natural\rangle \times id}\downarrow & & \downarrow{\scriptstyle \langle \pi_1, t^\natural\rangle} \\
(a \times (b \Rightarrow T)) \times b & \xrightarrow[\ \pi_1\ ]{} & a \times (b \Rightarrow T)
\end{array}$$

is a pull-back diagram induced by first projections. Hence, (5.22) is equivalent to

$$\phi \ \sqsubseteq\ \forall[\pi_1]\mathbf{I}[app \circ (\langle \pi_1, t^\natural\rangle \times id)]\psi$$

which in turn is equivalent to

$$\mathbf{I}[\pi_1]\phi \;\sqsubseteq\; \mathbf{I}[app\circ(\langle\pi_1, t^\sharp\rangle \times id)]\psi \tag{5.23}$$

since $\forall[\pi_1]$ is right adjoint to $\mathbf{I}[\pi_1]$. Analyzing the right side of (5.23) one finds that $app\circ(\langle\pi_1, t^\sharp\rangle \times id) = \langle\pi_1, t\rangle \circ flip \circ swap$, so that (5.23) becomes

$$\mathbf{I}[\pi_1]\phi \;\sqsubseteq\; \mathbf{I}[flip\circ swap]\mathbf{I}[\langle\pi_1, t\rangle]\psi$$

which immediately follows from the assumption (5.21) if the monotone map $\mathbf{I}[flip\circ swap]$ is applied to both sides, and noting that $(\pi_1 \times id)\circ flip\circ swap = \pi_1$.

Thus, we have shown that both maps $s \mapsto s^\flat$ and $t \mapsto t^\sharp$ are well-defined. The claim is that they are mutually inverse and natural in objects (S, ϕ) and (T, ψ). The first part is expressed by the equations

$$(t^\sharp)^\flat = t \qquad\qquad (s^\flat)^\sharp = s$$

and the second part by

$$\begin{aligned} (s\circ h)^\flat &= s^\flat \circ \mathbf{DI}[\pi_1]h & (h\circ t)^\sharp &= (\Pi[\pi_1]h)\circ t^\sharp \\ (t\circ(\mathbf{DI}[\pi_1]))^\sharp &= t^\sharp \circ h & ((\Pi[\pi_1]g)\circ s)^\flat &= g\circ s^\flat. \end{aligned}$$

Note, of the latter four equations only one of the two rows needs to be proven as the other follows because $(t^\sharp)^\flat = t$ and $(s^\flat)^\sharp = s$ for all t, s. Of course, all equations are to be read as equations in $\mathbf{DI}[a]$ or $\mathbf{DI}[a \times b]$, so that *e.g.* $(s\circ h)^\flat = s^\flat \circ \mathbf{DI}[\pi_1]h$ really means

$$eval\circ(id \times (s\circ\langle\pi_1, h\rangle))\circ swap \;=\; eval\circ(id \times s)\circ swap\circ\langle\pi_1, h\circ(\pi_1 \times id)\rangle.$$

All equations are readily verified and proofs are omitted. This completes the proof that the functor $\Pi[\pi_1] : \mathbf{DI}[a \times b] \to \mathbf{DI}[a]$ is right adjoint to $\mathbf{DI}[\pi_1]$.

Finally, it is shown that $\Pi[\pi_1]$ satisfies Beck-Chevalley for pull-back squares of the form

$$\begin{array}{ccc} a \times b & \xrightarrow{\pi_1} & a \\ {\scriptstyle t\times id}\downarrow & & \downarrow{\scriptstyle t} \\ a' \times b & \xrightarrow[\pi_1]{} & a' \end{array}$$

where $t : a \to a'$ morphism in $\mathcal{C}$, *i.e.* that we have $\mathbf{DI}[t] \circ \Pi[\pi_1] \cong \Pi[\pi_1] \circ \mathbf{DI}[t \times id]$. This is an equivalence between functors, so we must check an object and a morphism part. As to objects we find $(\mathbf{DI}[t] \circ \Pi[\pi_1])(S, \phi) = (b \Rightarrow S, \mathbf{I}[t \times id]\forall[\pi_1]\mathbf{I}[app]\phi)$ and $(\Pi[\pi_1] \circ \mathbf{DI}[t \times id])(S, \phi) = (b \Rightarrow S, \forall[\pi_1]\mathbf{I}[app]\mathbf{I}[(t \times id) \times id]\phi)$ which indeed are equivalent, for

$$\begin{aligned}
&\mathbf{I}[t \times id]\forall[\pi_1]\mathbf{I}[app] \\
&\quad \cong \forall[\pi_1]\mathbf{I}[(t \times id) \times id]\mathbf{I}[app] \\
&\quad = \forall[\pi_1]\mathbf{I}[app \circ ((t \times id) \times id)] \\
&\quad = \forall[\pi_1]\mathbf{I}[((t \times id) \times id) \circ app] \\
&\quad = \forall[\pi_1]\mathbf{I}[app]\mathbf{I}[(t \times id) \times id]
\end{aligned}$$

where the first line is a consequence of Beck-Chevalley for $\mathbf{I}$. As to morphisms we compute

$$\begin{aligned}
&(\mathbf{DI}[t] \circ \Pi[\pi_1])f \\
&\quad = \mathbf{DI}[t]curry(f \circ app) \\
&\quad = curry(f \circ app) \circ (t \times id) \\
&\quad = curry(f \circ app \circ ((t \times id) \times id)) \\
&\quad = curry(f \circ ((t \times id) \times id) \circ app) \\
&\quad = \Pi[\pi_1](f \circ ((t \times id) \times id)) \\
&\quad = (\Pi[\pi_1]\mathbf{DI}[t \times id])f.
\end{aligned}$$

∎

Theorem 5.2.14 *$\mathcal{DB}_= = (\mathcal{T}_=, \mathbf{DB}_= : \mathcal{T}_=^{op} \to \mathbf{Cat})$ is a hyperdoctrine.*

Proof: by Theorems 5.2.9 and 5.2.13. ∎

Remark: Theorem 5.2.13 requires that the products $\times$ of the cartesian closed structure of $\mathcal{T}_=$ and the products that determine the construction of $\mathcal{DB}_=$ are

the same. From now on we will assume, unless stated otherwise, that these products are the $\mathcal{T}_0$-products. Thus, for $\sigma = \sigma_1, \ldots, \sigma_m$ object in $\mathcal{T}_=$, an object in $\mathbf{DB}_=[\sigma]$ is a pair $(\tau, [w \vdash \phi])$ with $\tau = \tau_1, \ldots, \tau_n$ and w variable of type $(\sigma_1 \times \cdots \times \sigma_m) \times (\tau_1 \times \cdots \times \tau_n)$.

The last theorem says that the structure of $\mathcal{DB}_=$ is rich enough to interpret the ordinary (first-order) predicate logical part of $\mathcal{L}$. Also we know how to interpret the atomic formulae, *i.e.* formulae of form $\iota\phi$ where ϕ is a proposition of the base logic: we take the object in $\mathcal{B}_=$ corresponding to ϕ and translate it into $\mathcal{DB}_=$ via the (full and faithful) embedding $\iota : \mathcal{B}_= \dot{\to} \mathcal{DB}_=$ defined in Lemma 5.2.11. What is missing still is the interpretation of the modal operator $\Diamond$. For the particular notion of constraint $(\Omega^*, [\,], @)$ the idea is to get the identification

$$\Diamond M \;\cong\; \Sigma c^{\Omega^*}.\, \iota(\sqcap c) \supset M$$

where $\sqcap$ is the function that conjoins the propositions in the list c into a single proposition $\sqcap c$. Let us translate this into more general terms so that it may be applied to other notions of constraints as well.

Definition 5.2.15 *Let $\mathcal{I} = (\mathcal{C}, \mathbf{I} : \mathcal{C}^{op} \to \mathbf{PreOrd})$ be an indexed preorder with finite products $(\wedge, true)$ in the fibres. A notion of* constraint *on $\mathcal{I}$ is an object $\mathbf{c}$ in $\mathcal{C}$ together with*

- *a map that assigns to each morphism $c : a \to \mathbf{c}$ and element ϕ in $\mathbf{I}[a]$ an element ϕ^c in $\mathbf{I}[a]$*
- *a morphism $e_a : a \to \mathbf{c}$ for each a in $\mathcal{C}$*
- *a map that assigns to each pair of morphisms $f, g : a \to \mathbf{c}$ a morphism $f \cdot g : a \to \mathbf{c}$*

subject to the following conditions:

1. $e_a \circ t = e_b$ *and* $(f \cdot g) \circ t = (f \circ t) \cdot (g \circ t)$

2. $f, g : a \to \mathbf{c}$, *then* $\phi^{e_a} = \phi$ *and* $\phi^{f \cdot g} = (\phi^f)^g$

3. $e_a \cdot f = f = f \cdot e_a$ *and* $(f \cdot g) \cdot h = f \cdot (g \cdot h)$

4. *If* $\xi \wedge \phi \sqsubseteq \psi$, *then* $\xi \wedge \phi^f \sqsubseteq \psi^f$, *and* $true \sqsubseteq true^f$

5. $\mathbf{I}[t](\phi^f) = (\mathbf{I}[t]\phi)^{f \circ t}$

where $f, g, h : a \to \mathbf{c}$, $t : b \to a$, *and* ϕ, ψ, ξ *objects in* $\mathbf{I}[a]$. *The object* $\mathbf{c}$ *is the* constraint object. *A morphism* $f : a \to \mathbf{c}$ *is called a* constraint *over* a. *The morphism* $e_a : a \to \mathbf{c}$ *is called* vacuous *constraint over* a, *the index* a *is sometimes omitted. The combination* $f \cdot g$ *of two constraints is the* multiplication *of* f *and* g.

Definition 5.2.16 *Let* $\mathcal{I} = (\mathcal{C}, \mathbf{I} : \mathcal{C}^{op} \to \mathbf{PreOrd})$ *be an indexed preorder with finite products in the fibres and* $\mathcal{DI} = (\mathcal{C}, \mathbf{DI} : \mathcal{C}^{op} \to \mathbf{Cat})$ *as in Theorem 5.2.13. Given a notion of constraint on* $\mathcal{I}$ *one defines for each object* (S, ϕ) *in* $\mathbf{DI}[a]$ *an object* $\Diamond[a](S, \phi)$ *in* $\mathbf{DI}[a]$ *as follows*

$$\Diamond[a](S, \phi) \stackrel{df}{=} (\mathbf{c} \times S, (\mathbf{I}[id \times \pi_2]\phi)^{\pi_1 \circ \pi_2})$$

where π_2 *is the projection* $\mathbf{c} \times S \to S$ *or* $a \times (\mathbf{c} \times S) \to \mathbf{c} \times S$ *as appropriate.*

Before we analyze the properties of the map $(S, \phi) \mapsto \Diamond[a](S, \phi)$ in general, let us concretize it to the special situation of $(\Omega^*, [\,], @)$ as the notion of constraint and the indexed category $\mathcal{DB}_=$.

Lemma 5.2.17 *The following data make up a notion of constraint on* $\mathcal{B}_= = (\mathcal{T}_=, \mathbf{B}_= : \mathcal{T}_=^{op} \to \mathbf{PreOrd})$ *where the finite products* $(\wedge, true)$ *in the fibres* $\mathbf{B}_=[\tau]$ *are the ones defined in the proof of Theorem 5.2.9:*

- *Constraint object in* $\mathcal{T}_=$ *is* Ω^*
- *Given* $[\Delta \vdash f] : \tau \to \Omega^*$ *in* $\mathcal{T}_=$ *and* $[\Delta \vdash \phi]$ *in* $\mathbf{B}_=[\tau]$, *then put*

$$[\Delta \vdash \phi]^{[\Delta \vdash f]} \stackrel{df}{=} [\Delta \vdash \phi^f]$$

where ϕ^f *is weak* f ϕ.

- *Vacuous constraint over τ is* $[\Delta \vdash [\,]]$
- *Multiplication of* $[\Delta \vdash f] : \tau \to \Omega^*$ *and* $[\Delta \vdash g] : \tau \to \Omega^*$ *is*

$$[\Delta \vdash f] \cdot [\Delta \vdash g] \quad \stackrel{df}{=} \quad [\Delta \vdash f \mathbin{@} g]$$

where τ object in $\mathcal{T}_=$ and Δ context such that $\|\Delta\| = \tau$.

Proof: The definition of multiplication of constraints and of $[\Delta_\phi \vdash \phi]^{[\Delta_f \vdash f]}$ is independent of the choice of representatives. The conditions (1)–(5) of Definition 5.2.15 immediately follow from Lemma 3.1.7 and 3.1.8 and the properties of substitution. ∎

Remark: The construction in Lemma 5.2.17 can be generalized in the apparent way to any (syntactic) notion of constraint $(\mathbf{C}, 1, \cdot)$ with action map $c, \phi \mapsto \phi^c$ in the sense of Definition 3.1.9. We will refer to the notion of constraint in Lemma 5.2.17 by the triple $(\Omega^*, [\,], @)$.

Lemma 5.2.18 *Let τ be object in $\mathcal{T}_=$ and M object in $\mathbf{DB}_=[\tau]$. Further, let the object $\Diamond[\tau]M$ in $\mathbf{DB}_=[\tau]$ be defined according to Definition 5.2.16 for the specific notion of constraint $(\Omega^*, [\,], @)$ (cf. Lemma 5.2.17). Then,*

$$\Diamond[\tau]M \quad \cong \quad \Sigma[\pi_1]((\iota \sqcap_\tau) \supset \mathbf{DB}_=[\pi_1]M)$$

where

$$\sqcap_\tau \quad \stackrel{df}{=} \quad [\Delta_\tau, z^{\Omega^*} \vdash \sqcap z : \Omega]$$

is an object in $\mathbf{B}_=[\tau \times \Omega^*]$, Δ_τ an arbitrary context such that $\|\Delta_\tau\| = \tau$, and z an arbitrary variable not occurring in Δ_τ. ι is the embedding functor $\iota : \mathbf{B}_=[\tau \times \Omega^*] \to \mathbf{DB}_=[\tau \times \Omega^*]$. $\supset$ is exponentiation in $\mathbf{DB}_=[\tau \times \Omega^*]$, and $\pi_1 : \tau \times \Omega^* \to \tau$ first projection.

Proof: We will make an exception and assume for the proof that the $\mathcal{T}$-products are used in the construction of $\mathbf{DB}_{=}[\tau]$ and of the left adjoints Σ and exponents $\supset$ in $\mathbf{DB}_{=}$.

Let M be an object in $\mathbf{DB}_{=}[\tau]$, say

$$M \;=\; (\sigma, [\Delta_\tau, \Delta_\sigma \vdash \phi])$$

with σ object in $\mathcal{T}_{=}$, Δ_τ and Δ_σ disjoint contexts such that $\| \Delta_\tau \| = \tau$ and $\|\Delta_\sigma\| = \sigma$. In general σ will be a list $\sigma = \sigma_1, \ldots, \sigma_n$ of types but for simplicity we assume here, without loss of generality, that $n = 1$ and $\Delta_\sigma = y$.

A tedious but straightforward process of applying definitions shows that the purported isomorphism

$$\Sigma[\pi_1]((\iota \sqcap_\tau) \supset \mathbf{DB}_{=}[\pi_1]M) \;\cong\; \Diamond[\tau]M$$

ultimately comes down to an isomorphism

$$(\Omega^* \times (1 \Rightarrow \sigma)\,,\, A) \;\cong\; (\Omega^* \times \sigma\,,\, B)$$

in $\mathbf{DB}_{=}[\tau]$, where

$$\begin{aligned} A &= [\Delta_\tau, z, w \vdash \forall v.\,(\sqcap z) \supset (\phi\{w\,v/y\})] \\ B &= [\Delta_\tau, z, y \vdash \phi^z]. \end{aligned}$$

Thus, we need morphisms

$$\begin{aligned} f &: \tau \times (\Omega^* \times (1 \Rightarrow \sigma)) \to \Omega^* \times \sigma \\ g &: \tau \times (\Omega^* \times \sigma) \to \Omega^* \times (1 \Rightarrow \sigma) \end{aligned}$$

which are mutually inverse and further is must hold

$$A \sqsubseteq \mathbf{B}_{=}[\langle \pi_1, f \rangle]\, B \qquad B \sqsubseteq \mathbf{B}_{=}[\langle \pi_1, g \rangle]\, A.$$

It is not difficult to check that the morphisms

$$\begin{aligned} f &\stackrel{df}{=} [\Delta_\tau, z, w \vdash z, w*] \\ g &\stackrel{df}{=} [\Delta_\tau, z, y \vdash z, \lambda v.\, y] \end{aligned}$$

do the job, where w is a variable of type $\mathbf{1} \Rightarrow \sigma$ and v a variable of type $\mathbf{1}$ both not occurring in Δ_τ, z, y. That f, g are inverse, *i.e.* $f \circ \langle \pi_1, g \rangle = id$ and $g \circ \langle \pi_1, f \rangle = id$ follows from β, η and ξ equalities for function type $\mathbf{1} \Rightarrow \sigma$ in $\mathcal{B}$. The inequality $A \sqsubseteq \mathbf{B}_{=}[\langle \pi_1, f \rangle] B$ amounts to deriving the sequent

$$\forall v.(\sqcap z) \supset (\phi\{w\, v/y\}) \;\; \vdash_{\Delta_\tau, z, w} \;\; (\phi^z)\{z/z\}\{w * /y\}$$

and the inequation $B \sqsubseteq \mathbf{B}_{=}[\langle \pi_1, g \rangle] A$ to deriving

$$\phi^z \;\; \vdash_{\Delta_\tau, z, y} \;\; (\forall v.(\sqcap z) \supset (\phi\{w\, v/y\}))\{z/z\}\{\lambda v.\, y/w\}$$

in $\mathcal{B}$, which are both readily obtained. Both derivations use the equivalence $\phi^z \cong (\sqcap z) \supset \phi$ and the former also the identity $(\phi^c)\{t/x\} = (\phi\{t/x\})^{c\{t/x\}}$. ∎

Theorem 5.2.19 *Let $\mathcal{I}$ and $\mathcal{DI}$ be as in Definition 5.2.16. Further assume that the finite products in the fibres of $\mathcal{I}$ are preserved by translations along morphisms in the base. Then, the mapping $M \mapsto \Diamond[a]M$ specified in Definition 5.2.16 induces a locally strong indexed monad on the indexed category $\mathcal{DI} = (\mathcal{C}, \mathbf{DI} : \mathcal{C}^{op} \to \mathbf{Cat})$. More specifically, for every a in $\mathcal{C}$, $\Diamond[a]$ can be extended to become a functor*

$$\Diamond[a] : \mathbf{DI}[a] \to \mathbf{DI}[a]$$

and there are natural transformations

$$\eta[a] : id \dot{\to} \Diamond[a] \qquad \mu[a] : \Diamond[a] \circ \Diamond[a] \dot{\to} \Diamond[a]$$

where id is the identity functor on $\mathbf{DI}[a]$, such that $(\Diamond[a], \eta[a], \mu[a])$ is a monad *on $\mathbf{DI}[a]$; Further, these monads are* natural *in the object a of $\mathcal{C}$, i.e. for every morphism $t : a \to b$ in $\mathcal{C}$ the equations*

$$\begin{aligned} \Diamond[a] \circ \mathbf{DI}[t] &= \mathbf{DI}[t] \circ \Diamond[b] \\ \eta[a]\mathbf{DI}[t] &= \mathbf{DI}[t]\eta[b] \\ \mu[a]\mathbf{DI}[t] &= \mathbf{DI}[t]\mu[b] \end{aligned}$$

hold, where the first is an identity of functors and the others identities of natural transformations. Finally, for every a the monad $(\Diamond[a], \eta[a], \mu[a])$ is strong, *i.e. there is a family of morphisms*

$$\sigma_{M,N} \;:\; M \wedge \Diamond[a]N \to \Diamond[a](M \wedge N)$$

in $\mathbf{DI}[a]$ *natural in M and N such that*

$$\begin{aligned}
(\Diamond[a]r_M) \circ \sigma_{true,M} &= r_{\Diamond[a]M} \\
(\Diamond[a]\alpha_{M,N,K}) \circ \sigma_{M\wedge N,K} &= \sigma_{M,N\wedge K} \circ (id_M \wedge \sigma_{N,K}) \circ \alpha_{M,N,\Diamond[a]K} \\
\sigma_{M,N} \circ (id_M \wedge \eta_N) &= \eta_{M\wedge N} \\
\sigma_{M,N} \circ (id_M \wedge \mu_N) &= \mu_{M\wedge N} \circ (\Diamond[a]\sigma_{M,N}) \circ \sigma_{M,\Diamond[a]N}
\end{aligned}$$

where r and α are the natural isomorphisms

$$\begin{aligned}
r_M &\;:\; true \wedge M \to M \\
\alpha_{M,N,K} &\;:\; (M \wedge N) \wedge K \to M \wedge (N \wedge K)
\end{aligned}$$

in $\mathbf{DI}[a]$.

Remark: We have to be careful to read the last four equations in the theorem as equations in $\mathbf{DI}[a]$: So, $\circ$ is composition in $\mathbf{DI}[a]$, and $\wedge$ is the product functor over $\mathbf{DI}[a]$ induced by the products in $\mathbf{I}[a]$. Composition $\circ$ was defined in Definition 5.2.10 and product functor $\wedge$ is understood to be determined as in the proof of Lemma 5.2.13.

We remark that the naturality of the monads $(\Diamond[a], \eta[a], \mu[a])$ in index a turn $\Diamond$ into a monad on $\mathbf{DI}$ in the 2-category of indexed categories [KS74].

Strong monads were shown to be a very useful concept for modelling various notions of computation [Mog89]. Here we may interpret $\Diamond$ as a particular notion of computation on *proofs*, *viz.* the computation of constraint information. The notion of a strong monad is explained in [Mog89]. There the morphisms σ are called *tensorial strength* of $\Diamond$.

Proof: The proof will make use of (1)–(5) in Definition 5.2.15 of a notion of constraint. We will refer to each of these as 'property (x)', where x is the appropriate number.

Fix an object a in $\mathcal{C}$. For ease of notation we write $\Diamond$ rather than $\Diamond[a]$ as the index a is understood. Let $\Diamond : \mathbf{DI}[a] \to \mathbf{DI}[a]$ be defined on objects as in Definition 5.2.16. It is extended to morphisms as follows:

$$\begin{array}{ccc} (S,\phi) & & (\mathbf{c}\times S, (\mathbf{I}[id\times\pi_2]\phi)^{\pi_1\circ\pi_2}) \\ f\downarrow & \mapsto & \downarrow(\pi_1\circ\pi_2, f\circ(id\times\pi_2)) \\ (T,\psi) & & (\mathbf{c}\times T, (\mathbf{I}[id\times\pi_2]\psi)^{\pi_1\circ\pi_2}) \end{array}$$

The morphism $\Diamond f = \langle \pi_1\circ\pi_2, f\circ(id\times\pi_2)\rangle$ in $\mathcal{C}$ has domain $a\times(\mathbf{c}\times S)$ and codomain $\mathbf{c}\times T$, so for $\Diamond f$ to be a well-defined morphism in $\mathbf{DI}[a]$ from $\Diamond(S,\phi) = (\mathbf{c}\times S, (\mathbf{I}[id\times\pi_2]\phi)^{\pi_1\circ\pi_2})$ to $\Diamond(T,\psi) = (\mathbf{c}\times T, (\mathbf{I}[id\times\pi_2]\psi)^{\pi_1\circ\pi_2})$ we must have

$$(\mathbf{I}[id\times\pi_2]\phi)^{\pi_1\circ\pi_2} \sqsubseteq \mathbf{I}[\langle\pi_1,\Diamond f\rangle](\mathbf{I}[id\times\pi_2]\psi)^{\pi_1\circ\pi_2}. \qquad (5.24)$$

Now, f is a morphism from (S,ϕ) to (T,ψ), so by definition $\phi \sqsubseteq \mathbf{I}[\langle\pi_1,f\rangle]\psi$ which implies $\mathbf{I}[id\times\pi_2]\phi \sqsubseteq \mathbf{I}[id\times\pi_2]\mathbf{I}[\langle\pi_1,f\rangle]\psi = \mathbf{I}[\langle\pi_1,f\rangle\circ(id\times\pi_2)]\psi = \mathbf{I}[\langle\pi_1, f\circ(id\times\pi_2)\rangle]\psi$. From property (4) we infer that $\delta\sqsubseteq\gamma$ implies $\delta^c\sqsubseteq\gamma^c$ for all δ,γ,c, so we obtain

$$(\mathbf{I}[id\times\pi_2]\phi)^{\pi_1\circ\pi_2} \sqsubseteq (\mathbf{I}[\langle\pi_1, f\circ(id\times\pi_2)\rangle]\psi)^{\pi_1\circ\pi_2}$$

which is the same as (5.24) for the right side can be transformed as follows

$$\begin{aligned} (\mathbf{I}[\langle\pi_1, f\circ(id\times\pi_2)\rangle]\psi)^{\pi_1\circ\pi_2} &= (\mathbf{I}[\langle\pi_1,\pi_2\circ\Diamond f\rangle])^{\pi_1\circ\Diamond f} \\ &= (\mathbf{I}[\langle\pi_1,\Diamond f\rangle]\mathbf{I}[id\times\pi_2])^{\pi_1\circ\pi_2\circ\langle\pi_1,\Diamond f\rangle} \\ &= \mathbf{I}[\langle\pi_1,\Diamond f\rangle](\mathbf{I}[id\times\pi_2]\psi)^{\pi_1\circ\pi_2} \end{aligned}$$

using property (5). Thus, we have shown (5.24), whence $\Diamond$ is well-defined on morphisms. One may check that the map $\Diamond$ actually defines a functor $\Diamond : \mathbf{DI}[a]\to\mathbf{DI}[a]$.

For $\Diamond$ to be a monad on $\mathbf{DI}[a]$ we need natural transformations $\eta : id \dot{\to} \Diamond$ and $\mu : \Diamond \circ \Diamond \dot{\to} \Diamond$ satisfying certain coherence equations. The element $\eta_{(S,\phi)} : (S,\phi) \to \Diamond(S,\phi)$ of η at object (S,ϕ) is defined as the morphism

$$\eta_{(S,\phi)} \stackrel{df}{=} e \times id \; : \; a \times S \to \mathbf{c} \times S$$

where e is the vacuous constraint at a. This is a well-defined morphism in $\mathbf{DI}[a]$ with domain (S,ϕ) and codomain $\Diamond(S,\phi)$ since

$$\begin{aligned} \phi &= \phi^e = (\mathbf{I}[id]\phi)^{e \circ \pi_1} \\ &= (\mathbf{I}[(id \times \pi_2) \circ \langle \pi_1, \eta_{(S,\phi)} \rangle]\phi)^{\pi_1 \circ \pi_2 \circ \langle \pi_1, \eta_{(S,\phi)} \rangle} \\ &= \mathbf{I}[\langle \pi_1, \eta_{(S,\phi)} \rangle](\mathbf{I}[id \times \pi_2]\phi)^{\pi_1 \circ \pi_2} \end{aligned}$$

by properties (2), (1), and (5). For arbitrary $f : (S,\phi) \to (T,\psi)$ it one verifies that $\Diamond f \circ \eta_{(S,\phi)} = \eta_{(T,\psi)} \circ f$ holds in $\mathbf{DI}[a]$, thus η is a natural transformation. The element $\mu_{(S,\phi)} : \Diamond \circ \Diamond(S,\phi) \to \Diamond(S,\phi)$ of μ at (S,ϕ) is defined as the morphism

$$\begin{aligned} \mu_{(S,\phi)} \stackrel{df}{=} & \langle (\pi_1 \circ \pi_2 \circ \pi_2) \cdot (\pi_1 \circ \pi_2), \pi_2 \circ \pi_2 \circ \pi_2 \rangle \\ : \; & a \times (\mathbf{c} \times (\mathbf{c} \times S)) \to \mathbf{c} \times S. \end{aligned}$$

We wish to show that $\mu_{(S,\phi)}$ is a well defined morphism with domain $\Diamond \circ \Diamond(S,\phi)$ and codomain $\Diamond(S,\phi)$. This comes down to proving the inequation

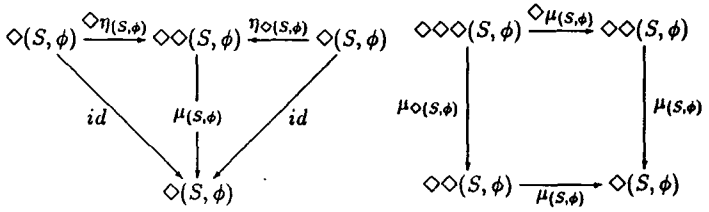
$$(\mathbf{I}[id \times \pi_2](\mathbf{I}[id \times \pi_2]\phi)^{\pi_1 \circ \pi_2})^{\pi_1 \circ \pi_2} \sqsubseteq \mathbf{I}[\langle \pi_1, \mu_{(S,\phi)} \rangle](\mathbf{I}[id \times \pi_2]\phi)^{\pi_1 \circ \pi_2}$$

Both sides are in fact equal which is seen as follows

$$\begin{aligned} &(\mathbf{I}[id \times \pi_2](\mathbf{I}[id \times \pi_2]\phi)^{\pi_1 \circ \pi_2})^{\pi_1 \circ \pi_2} \\ &\quad = ((\mathbf{I}[(id \times \pi_2) \circ (id \times \pi_2)]\phi)^{\pi_1 \circ \pi_2 \circ (id \times \pi_2)})^{\pi_1 \circ \pi_2} \\ &\quad = ((\mathbf{I}[id \times (\pi_2 \circ \pi_2)]\phi)^{\pi_1 \circ \pi_2 \circ \pi_2})^{\pi_1 \circ \pi_2} \\ &\quad = (\mathbf{I}[id \times (\pi_2 \circ \pi_2)]\phi)^{(\pi_1 \circ \pi_2 \circ \pi_2) \cdot (\pi_1 \circ \pi_2)} \\ &\quad = (\mathbf{I}[(id \times \pi_2) \circ \langle \pi_1, \mu_{(S,\phi)} \rangle]\phi)^{\pi_1 \circ \pi_2 \circ \langle \pi_1, \mu_{(S,\phi)} \rangle} \\ &\quad = \mathbf{I}[\langle \pi_1, \mu_{(S,\phi)} \rangle](\mathbf{I}[id \times \pi_2]\phi)^{\pi_1 \circ \pi_2} \end{aligned}$$

using properties (2) and (5). The proof of the naturality equation $\mu_{(T,\psi)} \circ (\Diamond \circ \Diamond f) = \Diamond f \circ \mu_{(S,\phi)}$ is omitted, We find that μ is a natural transformation.

Now we come to check the coherence equations for η and μ that make $(\Diamond, \eta, \mu)$ a monad. It is to be shown that the following diagrams commute in $\mathbf{DI}[a]$:



Proofs of these equations are obtained using the fact that multiplication of constraints is a monoid, *i.e.* by property (3).

To sum up, we have defined a monad $(\Diamond[a], \eta[a], \mu[a])$ for each object a in $\mathcal{C}$. Next we must prove that these monads are natural in a. To this end assume a morphism $t : a \to b$ in $\mathcal{C}$. $\Diamond[a]$ is natural in a if functors $\Diamond[a] \circ \mathbf{DI}[t]$ and $\mathbf{DI}[t] \circ \Diamond[b]$ are equal. For object (S, ϕ) in $\mathbf{DI}[b]$ we compute

$$\begin{aligned}
(\mathbf{DI}[t] \circ \Diamond[b])(S,\phi) &= \mathbf{DI}[t](\Diamond[b](S,\phi)) \\
&= \mathbf{DI}[t](\mathbf{c} \times S, (\mathbf{I}[id \times \pi_2]\phi)^{\pi_1 \circ \pi_2}) \\
&= (\mathbf{c} \times S, \mathbf{I}[t \times id](\mathbf{I}[id \times \pi_2]\phi)^{\pi_1 \circ \pi_2}) \\
&= (\mathbf{c} \times S, (\mathbf{I}[t \times id]\mathbf{I}[id \times \pi_2]\phi)^{\pi_1 \circ \pi_2 \circ (t \times id)}) \\
&= (\mathbf{c} \times S, (\mathbf{I}[t \times \pi_2]\phi)^{\pi_1 \circ \pi_2})
\end{aligned}$$

using property (5). For the other functor we compute

$$\begin{aligned}
(\Diamond[a] \circ \mathbf{DI}[t])(S,\phi) &= \Diamond[a](\mathbf{DI}[t](S,\phi)) \\
&= (\mathbf{c} \times S, (\mathbf{I}[id \times \pi_2]\mathbf{I}[t \times id]\phi)^{\pi_1 \circ \pi_2}) \\
&= (\mathbf{c} \times S, (\mathbf{I}[t \times \pi_2]\phi)^{\pi_1 \circ \pi_2})
\end{aligned}$$

Thus, functors $\Diamond[a] \circ \mathbf{DI}[t]$ and $\mathbf{DI}[t] \circ \Diamond[b]$ agree on objects. The proof that they agree on morphisms too, is similarly straightforward. Naturality of $\eta[a]$ in a

requires that $\eta[a]\mathbf{DI}[t]$ and $\mathbf{DI}[t]\eta[b]$ are equal as natural transformations, so we need to evaluate them only on objects M in $\mathbf{DI}[b]$:

$$\begin{aligned}(\eta[a]\mathbf{DI}[t])M &= \eta[a]_{\mathbf{DI}[t]M} = e_a \times id = (e_b \circ t) \times id \\ &= (e_b \times id) \circ (t \times id) = \mathbf{DI}[t](e_b \times id) \\ &= \mathbf{DI}[t]\eta[b]\end{aligned}$$

The third equation holds by property (1). The remaining proof that natural transformations $\mu[a]\mathbf{DI}[t]$ and $\mathbf{DI}[t]\mu[b]$ are equal, which also invokes property (1), is omitted.

Finally, the monads $\Diamond[a]$ are strong, *i.e.* there exists a family of morphisms

$$\sigma_{(S,\phi),(T,\psi)} : (S,\phi) \wedge \Diamond[a](T,\psi) \to \Diamond[a]((S,\phi) \wedge (T,\psi))$$

natural in (S,ϕ), (T,ψ) satisfying the four coherence equations stated in the theorem. We claim that such $\sigma_{(S,\phi),(T,\psi)}$ are given by

$$\begin{aligned}\sigma_{(S,\phi),(T,\psi)} &\stackrel{df}{=} \langle \pi_1 \circ \pi_2 \circ \pi_2, \langle \pi_1 \circ \pi_2, \pi_2 \circ \pi_2 \circ \pi_2 \rangle \rangle \\ &: a \times (S \times (\mathbf{c} \times T)) \to \mathbf{c} \times (S \times T).\end{aligned}$$

For these morphisms to be well-defined as morphisms in $\mathbf{DI}[a]$ we need to verify the inequation

$$\mathbf{I}[id \times \pi_1]\phi \wedge \mathbf{I}[id \times \pi_2](\mathbf{I}[id \times \pi_2]\psi)^{\pi_1 \circ \pi_2} \tag{5.25}$$

$$\sqsubseteq \mathbf{I}[\langle \pi_1, \sigma \rangle](\mathbf{I}[id \times \pi_2](\mathbf{I}[id \times \pi_1]\phi \wedge \mathbf{I}[id \times \pi_2]\psi))^{\pi_1 \circ \pi_2} \tag{5.26}$$

where from now on the indices of σ are dropped. Using property (5) the left side is shown to be equal to $\mathbf{I}[id \times \pi_1]\phi \wedge (\mathbf{I}[id \times (\pi_2 \circ \pi_2)]\psi)^{\pi_1 \circ \pi_2 \circ \pi_2}$ and the right side is simplified as follows:

$$\begin{aligned}&\mathbf{I}[\langle \pi_1, \sigma \rangle](\mathbf{I}[id \times \pi_2](\mathbf{I}[id \times \pi_1]\phi \wedge \mathbf{I}[id \times \pi_2]\psi))^{\pi_1 \circ \pi_2} \\ &\quad \cong \mathbf{I}[\langle \pi_1, \sigma \rangle](\mathbf{I}[id \times (\pi_1 \circ \pi_2)]\phi \wedge \mathbf{I}[id \times (\pi_2 \circ \pi_2)]\psi)^{\pi_1 \circ \pi_2} \\ &\quad = (\mathbf{I}[\langle \pi_1, \sigma \rangle](\mathbf{I}[id \times (\pi_1 \circ \pi_2)]\phi \wedge \mathbf{I}[id \times (\pi_2 \circ \pi_2)]\psi))^{\pi_1 \circ \pi_2 \circ \langle \pi_1, \sigma \rangle} \\ &\quad \cong (\mathbf{I}[\langle \pi_1, \sigma \rangle]\mathbf{I}[id \times (\pi_1 \circ \pi_2)]\phi \wedge \mathbf{I}[\langle \pi_1, \sigma \rangle]\mathbf{I}[id \times (\pi_2 \circ \pi_2)]\psi)^{\pi_1 \circ \sigma} \\ &\quad = (\mathbf{I}[id \times \pi_1]\phi \wedge \mathbf{I}[id \times (\pi_2 \circ \pi_2)]\psi)^{\pi_1 \circ \pi_2 \circ \pi_2}.\end{aligned}$$

The first equation is due to property (5) and the equivalences $\cong$ due to the assumption that translations $\mathbf{I}[\cdot]$ preserve products $\wedge$. Thus, we find that inequation (5.26) is equivalent to

$$\begin{aligned}\mathbf{I}[id \times \pi_1]\phi \ \wedge \ (\mathbf{I}[id \times (\pi_2 \circ \pi_2)]\psi)^{\pi_1 \circ \pi_2 \circ \pi_2} \\ \sqsubseteq \ (\mathbf{I}[id \times \pi_1]\phi \wedge \mathbf{I}[id \times (\pi_2 \circ \pi_2)]\psi)^{\pi_1 \circ \pi_2 \circ \pi_2}\end{aligned}$$

But this follows immediately from property (4). Namely, for arbitrary ξ, ϕ we have $\xi \wedge \phi \sqsubseteq \xi \wedge \phi$, hence by property (4), $\xi \wedge \phi^c \sqsubseteq (\xi \wedge \phi)^c$. Thus we have verified that the morphisms

$$\sigma_{(S,\phi),(T,\psi)} : (S,\phi) \wedge \Diamond[a](T,\psi) \to \Diamond[a]((S,\phi) \wedge (T,\psi))$$

are well-defined. Now, we wish to show that they are natural in (S,ϕ) and (T,ψ). This means we assume morphisms $f : (S,\phi) \to (S',\phi')$ and $g : (T,\psi) \to (T',\phi')$ in $\mathbf{DI}[a]$ and verify the identities

$$\begin{aligned}\Diamond[a](f \wedge id) \circ \sigma_{(S,\phi),(T,\psi)} &= \sigma_{(S',\phi'),(T,\psi)} \circ (f \wedge id) \\ \Diamond[a](id \wedge g) \circ \sigma_{(S,\phi),(T,\psi)} &= \sigma_{(S,\phi),(T',\psi')} \circ (id \wedge \Diamond g)\end{aligned}$$

where $\circ$ and $\wedge$ are composition and product functor in $\mathbf{DI}[a]$. When definitions are unrolled, then these two equations become

$$\begin{aligned}&\langle \pi_{12}, \langle f \circ \langle \pi_1, \pi_{12} \rangle, \pi_{22} \rangle \circ (id \times \pi_2) \rangle \ \circ \ \langle \pi_{122}, \langle \pi_{12}, \pi_{222} \rangle \rangle \\ &\quad = \ \langle \pi_{122}, \langle \pi_{12}, \pi_{222} \rangle \rangle \circ \langle \pi_1, \langle f \circ \langle \pi_1, \pi_{12} \rangle, \pi_{22} \rangle \rangle \\ &\langle \pi_{12}, \langle \pi_{12}, g \circ \langle \pi_1, \pi_{22} \rangle \rangle \circ (id \times \pi_2) \rangle \ \circ \ \langle \pi_1, \langle \pi_{122}, \langle \pi_{12}, \pi_{222} \rangle \rangle \rangle \\ &\quad = \ \langle \pi_{122}, \langle \pi_{12}, \pi_{222} \rangle \rangle \circ \langle \pi_1, \langle \pi_{12}, \langle \pi_{12}, g \circ (id \times \pi_2) \rangle \circ \langle \pi_1, \pi_{22} \rangle \rangle \rangle\end{aligned}$$

where $\pi_{klm\ldots}$ stands for $\pi_k \circ \pi_l \circ \pi_m \circ \cdots$. These equations are trivial to check.

Thus, the $\sigma_{(S,\phi),(T,\psi)}$ are natural in both indices. To complete the proof of the theorem we need to verify the four coherence equations

$$\begin{aligned}(\Diamond[a]r_M) \circ \sigma_{true,M} &= r_{\Diamond[a]M} \\ (\Diamond[a]\alpha_{M,N,K}) \circ \sigma_{M \wedge N,K} &= \sigma_{M,N \wedge K} \circ (id_M \wedge \sigma_{N,K}) \circ \alpha_{M,N,\Diamond[a]K}\end{aligned}$$

$$\sigma_{M,N} \circ (id_M \wedge \eta_N) = \eta_{M \wedge N}$$
$$\sigma_{M,N} \circ (id_M \wedge \mu_N) = \mu_{M \wedge N} \circ (\Diamond[a]\sigma_{M,N}) \circ \sigma_{M,\Diamond[a]N}$$

assuming objects $M = (S, \phi)$ and (T, ψ) where $r_M : true \wedge M \to M$ and $\alpha_{M,N,K} : (M \wedge N) \wedge K \to M \wedge (N \wedge K)$ are the canonical isomorphisms in $\mathbf{DI}[a]$. As before, these equation are to be read in $\mathbf{DI}[a]$ taking the appropriate definitions of $\circ$ and $\wedge$, and the morphisms r and α are

$$r = \pi_{22} \qquad \alpha = \langle \pi_{112}, \langle \pi_{212}, \pi_{22} \rangle \rangle.$$

The proof of the equations, which are straightforward, are omitted.

We now come to the final result of this chapter linking up the category theoretical construction of hyperdoctrine $\mathcal{DB}_=$ with the extraction of constraint terms and constraint predicates from derivations in lax logic as presented in Chapter 3.2. It is shown that the natural interpretation of $\mathcal{L}$ in $\mathcal{DB}_=$, treating $\Diamond$ as the monad induced by the notion of constraint $(\Omega^*, [\,], @)$, corresponds precisely to the constraint extraction process of Chapter 3.2. In order to keep things simple we will deal only with the closed propositional fragment $\mathcal{L}_0$ of lax logic. We believe that the result can be extended to $\mathcal{L}$ to cover quantifiers and substitution.

We associate with every well-formed closed formula M of $\mathcal{L}_0$, *i.e.* $\vdash M$ **wff**, an object $[\![M]\!]$ in $\mathbf{DB}_=[()]$ according to the following schema

$$[\![\iota\phi]\!] \stackrel{df}{=} \iota[\vdash \phi] \tag{5.27}$$
$$[\![true]\!] \stackrel{df}{=} true \tag{5.28}$$
$$[\![false]\!] \stackrel{df}{=} false \tag{5.29}$$
$$[\![M \wedge N]\!] \stackrel{df}{=} [\![M]\!] \wedge [\![N]\!] \tag{5.30}$$
$$[\![M \vee N]\!] \stackrel{df}{=} [\![M]\!] \vee [\![N]\!] \tag{5.31}$$
$$[\![M \supset N]\!] \stackrel{df}{=} [\![M]\!] \supset [\![N]\!] \tag{5.32}$$
$$[\![\Diamond M]\!] \stackrel{df}{=} \Diamond[\![M]\!] \tag{5.33}$$

Remark: In case (5.27) since formula $\iota\phi$ is supposed to be well-formed and closed, ϕ must be a well-formed closed proposition, *i.e.* $\vdash \phi : \Omega$. In the definiens of case (5.27), then, $[\vdash\phi]$ stands for the equivalence class of ϕ in $\mathbf{B}_{=}[()]$ and ι is the embedding $\iota : \mathbf{B}_{=}[()] \to \mathbf{DB}_{=}[()]$.

The cases (5.28)–(5.32) deal with the propositional formulae of $\mathcal{L}$. Each propositional connective is interpreted by the corresponding categorical operation. So, *true*, *false*, $\wedge$, $\vee$, and $\supset$ are translated into terminal object, initial object, product, sum, and exponent, respectively, in $\mathbf{DB}_{=}[()]$.

The last case (5.33) is to be read as interpreting the modal operator $\Diamond$ by the monad $\Diamond : \mathbf{DB}_{=}[()] \to \mathbf{DB}_{=}[()]$ that is induced by the notion of constraint $(\Omega^*, [\,], @\,)$. The definition of this monad is given by Lemma 5.2.17 and Definition 5.2.16.

To reduce notation we write $\mathbf{DB}_{=}$ for $\mathbf{DB}_{=}[()]$ and $\mathbf{B}_{=}$ for $\mathbf{B}_{=}[()]$. The following definition will facilitate the access to objects and morphisms in $\mathbf{DB}_{=}$, which are pairs $(\sigma, [w \vdash \phi])$ with $\sigma = \sigma_n, \ldots, \sigma_1$, and w variable of type $() \times \sigma = () \times (\sigma_n \times \cdots \times \sigma_1)$, where $\times$ from now on stands for the $\mathcal{T}_0$ product. It will be convenient to break convention to let $\times$ associate to the left and to reverse the numbering of sequences to go from right to left.

Definition 5.2.20 *Let $[\Delta \vdash t]$ be a morphism $\sigma \to \tau$ in $\mathcal{T}_{=}$ with $\sigma = \sigma_n, \ldots, \sigma_1$, τ a single type, and $\Delta = x_n, \ldots, x_1$ such that $\|\Delta\| = \sigma$. From this we obtain a morphism $() \times \sigma \to \tau$ in the following way:*

$$[\Delta \vdash t]^* \stackrel{df}{=} [w \vdash t\{\pi_n^n(\pi_2 w)/x_n\} \cdots \{\pi_1^n(\pi_2 w)/x_1\}]$$

where w is an arbitrary variable of type $() \times (\sigma_n \times \cdots \times \sigma_1)$ and π_k^n denotes the k-th projection $\sigma_n \times \cdots \times \sigma_1 \to \sigma_k$, $1 \leq k \leq n$, i.e.

$$\pi_{k+1}^{n+1} z = \pi_k^n (\pi_1 z) \qquad \pi_1^{n+1} z = \pi_2 z \qquad \pi_1^1 z = z.$$

The case $n = 0$ is treated separately:

$$[\vdash t]^* \stackrel{df}{=} [w \vdash t]$$

where w variable of type $() \times ()$.

Remark: The definition of $[\Delta \vdash t]^*$ does not depend on the choice of variable w.

Theorem 5.2.21 *Let M be a well-formed closed formula, i.e. $\vdash M$ wff. Further, let $|M|$ and M^* be constraint type and constraint predicate of M as defined in Chap. 3.2. Then, for every variable z of type $|M|$ we have*

$$[\![M]\!] \;=\; (|M|\,,\, [z \vdash M^*\, z]^*)$$

Proof: First note, $M^*\, z$ is a well-formed proposition with a single free variable z and $[z \vdash M^*\, z]$ morphism $|M| \to \Omega$ in $\mathcal{T}_=$. We can form $[z \vdash M^*\, z]^*$ to get a morphism $() \times |M| \to \Omega$, which is at the same time an object in $\mathbf{B}_=[() \times |M|]$, whence $(|M|\,,\, [z \vdash M^*\, z]^*)$ is object in $\mathbf{DB}_=$. The identity

$$[\![M]\!] \;=\; (|M|\,,\, [z \vdash M^*\, z]^*)$$

is proven by induction on the structure of M. We will give proofs only for the cases (5.27), (5.30), and (5.33). The other cases, which can be obtained in a similar way, are omitted.

- (5.27) We want

$$[\![\iota\phi]\!] \;=\; (|\iota\phi|\,,\, [z \vdash (\iota\phi)^*\, z]^*) \qquad (5.34)$$

where variable z has type $|\iota\phi| = \mathbf{1}$. Plugging together definitions we compute $[\![\iota\phi]\!] = \iota[\vdash\phi] = (\mathbf{1}, \mathbf{B}_=[\pi_1][\vdash\phi])$ where π_1 is the first projection $() \times \mathbf{1} \to ()$, *i.e.* $\pi_1 = [w\vdash\,]$ with w variable of type $()\times\mathbf{1}$. Thus, $\mathbf{B}_=[\pi_1][\vdash\phi] = \mathbf{B}_=[[w\vdash\,]][\vdash\phi] = [w \vdash \phi] = [z \vdash \phi]^* = [z \vdash (\iota\phi)^*\, z]^*$. This proves (5.34).

- (5.30) We want

$$[\![M \wedge N]\!] \;=\; (|M| \times |N|\,,\, [z \vdash M^*(\pi_1 z) \wedge N^*(\pi_2 z)]^*) \qquad (5.35)$$

Unrolling the definitions and the inductional hypothesis yields

$$\begin{aligned}
[\![M \wedge N]\!] &= [\![M]\!] \wedge [\![N]\!] \\
&= (|M|, [z \vdash M^\# z]^*) \wedge (|N|, [z \vdash N^\# z]^*) \\
&= (|M| \times |N|, \mathbf{B}_=[id \times \pi_1][z \vdash M^\# z]^* \wedge \mathbf{B}_=[id \times \pi_2][z \vdash N^\# z]^*)
\end{aligned}$$

Here $id \times \pi_1$ is the canonical morphism $() \times (|M| \times |N|) \to () \times |M|$ in $\mathcal{T}_=$, *i.e.*

$$id \times \pi_1 = [w \vdash (\pi_1 w, \pi_1(\pi_2 w))]$$

Hence we get

$$\begin{aligned}
\mathbf{B}_=[id \times \pi_1][z \vdash M^\# z]^* &= \mathbf{B}_=[[w \vdash (\pi_1 w, \pi_1(\pi_2 w))]][w \vdash M^\#(\pi_2 w)] \\
&= [w \vdash M^\#(\pi_2(\pi_1 w, \pi_1(\pi_2 w)))] \\
&= [w \vdash M^\#(\pi_1(\pi_2 w))]
\end{aligned}$$

Notice that in the first line variable w is used with two different types. For the left $\vdash$, w has type $() \times (|M| \times |N|)$ while for the right $\vdash$ it has type $() \times |M|$. This confusion does no harm since both uses of w never occur in the same term. We will use w from now on as a generic variable in this sense. Similarly to above we have $\mathbf{B}_=[id \times \pi_2][z \vdash N^\# z]^* = [w \vdash N^\#(\pi_2(\pi_2 w))]$, whence

$$[\![M \wedge N]\!] = (|M| \times |N|, [w \vdash M^\#(\pi_1(\pi_2 w)) \wedge N^\#(\pi_2(\pi_2 w))])$$

which proves (5.35).

• (5.33) We want

$$[\![\Diamond M]\!] = (\Omega^* \times |M|, [z \vdash (M^\#(\pi_2 z))^{\pi_1 z}]^*) \tag{5.36}$$

The definition of the monad $\Diamond$ and the inductional hypothesis yield

$$\begin{aligned}
[\![\Diamond M]\!] &= \Diamond[\![M]\!] \\
&= \Diamond(|M|, [z \vdash M^\# z]^*) \\
&= (\Omega^* \times |M|, (\mathbf{B}_=[id \times \pi_2][z \vdash M^\# z]^*)^{\pi_1 \circ \pi_2})
\end{aligned}$$

where $id \times \pi_2$ is the canonical morphism $() \times (\Omega^* \times |M|) \to () \times |M|$, *i.e.* $id \times \pi_2 = [w \vdash (\pi_1 w, \pi_2(\pi_2 w))]$ and $\pi_1 \circ \pi_2$ is the canonical morphism $() \times (\Omega^* \times |M|) \to \Omega^*$, *i.e.* $\pi_1 \circ \pi_2 = [w \vdash \pi_1(\pi_2 w)]$. So, we can simplify further to get

$$\begin{aligned}
&(\mathbf{B}_=[id \times \pi_2][z \vdash M^* z]^*)^{\pi_1 \circ \pi_2} \\
&\quad = (\mathbf{B}_=[[w \vdash (\pi_1 w, \pi_2(\pi_2 w))]][w \vdash M^*(\pi_2 w)])^{[w \vdash \pi_1(\pi_2 w)]} \\
&\quad = [w \vdash M^*(\pi_2(\pi_2 w))]^{[w \vdash \pi_1(\pi_2 w)]} \\
&\quad = [w \vdash (M^*(\pi_2(\pi_2 w)))^{\pi_1(\pi_2 w)}]
\end{aligned}$$

where the last equation is due to the definition of the notion of constraint $(\Omega^*; [\,], @)$ on $\mathcal{B}_=$. This proves (5.36). ∎

The other part of constraint extraction in $\mathcal{L}$ is extracting from the derivation of a sequent

$$M_n, \ldots, M_1 \vdash_\Delta M$$

and sequence $z_n, \ldots, z_1$ of arbitrary fresh variables of types $|M_i|$, $i = 1, \ldots, n$, a well-formed constraint term

$$\Delta, z_n, \ldots, z_1 \vdash t : |M|$$

and a derivation of the sequent

$$M_n^* z_n, \ldots, M_1^* z_1 \vdash_{\Delta, z_n, \ldots, z_1} M^* t$$

in $\mathcal{B}$. We wish to show that this extraction process can be understood in terms of the categorical logic $\mathcal{DB}_=$ as interpreting derivations in $\mathcal{L}$ as morphisms in $\mathcal{DB}_=$.

Again, we focus on the closed propositional fragment of $\mathcal{L}$. We associate — rule by rule along the structure of the derivation tree — with every derivation in $\mathcal{L}_0$ a morphism such that

$$\Gamma \vdash M \;\mapsto\; m : [\![\Gamma]\!] \to [\![M]\!]$$

where $[\Gamma]$ is defined inductively as

$$[\Gamma, M_2, M_1] \stackrel{df}{=} [\Gamma, M_2] \wedge [M_1] \qquad [()] \stackrel{df}{=} \mathit{true}$$

with $\wedge$ and *true* the finite products in $\mathbf{DB}_=$, and prove that m is precisely the constraint term extracted from the derivation. This together with Theorem 5.2.21 provides a category theoretical semantics for $\mathcal{L}_0$ in $\mathcal{DB}_=$ which captures the interpretation of formulae as constraint types and constraint predicates and the interpretation of derivations as constraint terms.

The interpretation of $\mathcal{L}_0$'s structural rules in terms of morphisms in $\mathcal{DB}_=$ is given in Figure 5–1 and of the logical rules in Figure 5–2.

rule		*morphisms in* $\mathbf{DB}_=$
id	$M \vdash M$	$id : [M] \to [M]$
weak	$\dfrac{\Gamma, M \vdash N}{\Gamma \vdash N}$	$\dfrac{m \circ \pi_1 : [\Gamma] \wedge [M] \to [N]}{m : [\Gamma] \to [N]}$
perm	$\dfrac{\Gamma_1, N, M, \Gamma_2 \vdash K}{\Gamma_1, M, N, \Gamma_2 \vdash K}$	$\dfrac{m \circ t : (([\Gamma_1] \wedge [N]) \wedge [M]) \wedge [\Gamma_2] \to [K]}{m : (([\Gamma_1] \wedge [M]) \wedge [N]) \wedge [\Gamma_2] \to [K]}$ $t = \langle \langle \pi_1 \circ \pi_1, \pi_2 \rangle, \pi_2 \circ \pi_1 \rangle \times id$
cut	$\dfrac{\Gamma \vdash N}{\Gamma \vdash M \quad \Gamma, M \vdash N}$	$\dfrac{n \circ \langle id, m \rangle : [\Gamma] \to [N]}{m : [\Gamma] \to [M] \quad n : [\Gamma] \wedge [M] \to [N]}$

Figure 5–1: Interpretation of Structural Rules for $\mathcal{L}_0$

rule		morphisms in $\mathbf{DB}_{=}$
falseE	$\mathit{false} \vdash M$	$\Box : \mathit{false} \to [M]$
$\vee E$	$\dfrac{M \vee N \vdash K}{M \vdash K \quad N \vdash K}$	$\dfrac{[m,n] : [M] \vee [N] \to [K]}{m : [M] \to [K] \quad n : [N] \to [K]}$
$\vee I_l$	$M \vdash M \vee N$	$\iota_1 : [M] \to [M] \vee [N]$
$\vee I_r$	$N \vdash M \vee N$	$\iota_2 : [N] \to [M] \vee [N]$
trueI	$\vdash \mathit{true}$	$! : \mathit{true} \to \mathit{true}$
$\wedge I$	$\dfrac{\Gamma \vdash M \wedge N}{\Gamma \vdash M \quad \Gamma \vdash N}$	$\dfrac{\langle m,n \rangle : [\Gamma] \to [M] \wedge [N]}{m : [\Gamma] \to [M] \quad n : [\Gamma] \to [N]}$
$\wedge E_l$	$M \wedge N \vdash M$	$\pi_1 : [M] \wedge [N] \to [M]$
$\wedge E_r$	$M \wedge N \vdash N$	$\pi_2 : [M] \wedge [N] \to [N]$
$\supset I$	$\dfrac{\Gamma \vdash M \supset N}{\Gamma, M \vdash N}$	$\dfrac{\mathit{curry}(m) : [\Gamma] \to [M] \supset [N]}{m : [\Gamma] \wedge [M] \to [N]}$
$\supset E$	$\dfrac{\Gamma, M \vdash N}{\Gamma \vdash M \supset N}$	$\dfrac{\mathit{eval} \circ \langle \pi_2, m \circ \pi_1 \rangle : [\Gamma] \wedge [M] \to [N]}{m : [\Gamma] \to [M] \supset [N]}$
$\Diamond I$	$M \vdash \Diamond M$	$\eta : [M] \to \Diamond [M]$
$\Diamond M$	$\Diamond\Diamond M \vdash \Diamond M$	$\mu : \Diamond\Diamond [M] \to \Diamond [M]$
$\Diamond F$	$\dfrac{\Gamma, \Diamond M \vdash \Diamond N}{\Gamma, M \vdash N}$	$\dfrac{\Diamond m \circ \sigma : [\Gamma] \wedge \Diamond [M] \to \Diamond [N]}{m : [\Gamma] \wedge [M] \to [N]}$ $\sigma : [\Gamma] \wedge \Diamond [M] \to \Diamond([\Gamma] \wedge [M])$
ι	$\dfrac{\iota\phi \vdash \iota\psi}{\phi \vdash \psi}$	$\iota([\vdash \phi] \sqsubseteq [\vdash \psi]) : \iota[\vdash \phi] \to \iota[\vdash \psi]$

Figure 5–2: Interpretation of Logical Rules for $\mathcal{L}_0$

Remark: In Figures 5–1 and 5–2 it is assumed that the hypotheses lists Γ, Γ_1, Γ_2 are non-empty in which case we have $[\![\Gamma, M]\!] = [\![\Gamma]\!] \wedge [\![M]\!]$ and $[\![\Gamma_1, M, N, \Gamma_2]\!] = (([\![\Gamma_1]\!] \wedge [\![M]\!]) \wedge [\![N]\!]) \wedge [\![\Gamma_2]\!]$. This can be assumed without loss of generality since a dummy hypothesis *true* can always be introduced via rule *weak*. So, only for rule *weak* do we have to consider the special case where $\Gamma = ()$. In that case $[\![\Gamma]\!] = \mathit{true}$ and the interpretation of *weak* simplifies as follows

$$\frac{M \vdash N}{\vdash N} \quad \mapsto \quad \frac{m \circ !_{[\![M]\!]} : [\![M]\!] \to [\![N]\!]}{m : \mathit{true} \to [\![N]\!]}$$

where $!_{[\![M]\!]} : [\![M]\!] \to \mathit{true}$ is the terminal morphism in $\mathbf{DB}_=$.

Figure 5–2 needs more explanation: The morphism σ used in the translation of rule $\Diamond F$ is the tensorial strength of $\Diamond$. It is given by Theorem 5.2.19 as are η and μ in the translation of rules $\Diamond I$ and $\Diamond M$. Rule ι takes a sub-derivation in $\mathcal{B}$ of a sequent $\phi \vdash \psi$ and turns it into a derivation of $\iota\phi \vdash \iota\psi$ in $\mathcal{L}_0$. Its translation into $\mathbf{DB}_=$ is obtained as follows: Since ϕ, ψ are well-formed closed terms of type Ω we get well-defined objects $[\vdash \phi]$ and $[\vdash \psi]$ in $\mathbf{B}_=$. Since $\phi \vdash \psi$ is derivable we have a unique morphism $m : [\vdash \phi] \sqsubseteq [\vdash \psi]$ in $\mathbf{B}_=$ by definition of $\sqsubseteq$. This morphism is now mapped to the morphism

$$\iota m \ : \ \iota[\vdash \phi] \ \to \ \iota[\vdash \psi]$$

via the embedding functor $\iota : \mathbf{B}_= \to \mathbf{DB}_=$ defined in Lemma 5.2.11.

Observation 5.2.22 *The translation of Figs. 5–1 and 5–2 is total and correct, i.e. the translations can be applied rule by rule to any valid derivation in $\mathcal{L}_0$ of a sequent $\Gamma \vdash M$, and the result is a well-defined morphism $m : [\![\Gamma]\!] \to [\![M]\!]$ in $\mathbf{DB}_=$.*

Theorem 5.2.23 *Let a derivation of $\Gamma \vdash M$ in $\mathcal{L}_0$ with $\Gamma = M_n, \ldots, M_1$ be given and $m : [\![\Gamma]\!] \to [\![M]\!]$ be the corresponding morphism in $\mathbf{DB}_=$ obtained by Figures 5–1 and 5–2. Further, let $z_n, \ldots, z_1$ be arbitrary distinct variables of types $|M_i|$, $i = 1, \ldots, n$ respectively, and $z_n, \ldots, z_1 \vdash t : |M|$ the constraint term constructed by the translation of Figs. 3–10, 3–11 and 3–12 of Section 3.2. Then,*

$$m \ = \ [z_n, \ldots, z_1 \vdash t]^*$$

as morphisms in $\mathcal{T}_{=}$.

Proof: We will only give the proofs for rules *id*, *weak*, *falseE*, $\Diamond I$, $\Diamond M$ and $\Diamond F$. All other rules can be dealt with in a similar way. Before we start let us convince ourselves that the equation in the theorem makes sense: First suppose $n \geq 1$. By Theorem 5.2.21, the first component of $[\![\Gamma]\!]$ is the type $|M_n| \times \cdots \times |M_1|$ and the first component of $[\![M]\!]$ is $|M|$. Hence, $m : [\![\Gamma]\!] \to [\![M]\!]$ as a morphism in $\mathcal{T}_{=}$ has the same domain and codomain as the morphism $[z_n, \ldots, z_1 \vdash t]^*$. In the special case $n = 0$, *i.e.* $\Gamma = ()$, the first component of $[\![\Gamma]\!] = [\![()]\!] = \mathit{true}$ is $\mathbf{1}$, whence in this case, too, $m : [\![\Gamma]\!] \to [\![M]\!]$ has the same domain and codomain as $[\vdash t]^*$. Note that for the $\mathcal{T}_0$-product $() \times \mathbf{1} = \mathbf{1} \times \mathbf{1} = () \times ()$. The proof that $m = [z_n, \ldots, z_1 \vdash t]^*$ proceeds by induction on the structure of the derivation tree of $M_n, \ldots, M_1 \vdash M$.

- We begin with rule *id* which has the following translations:

$$M \vdash M \quad \mapsto \quad z^{|M|} \vdash z : |M| \qquad id : [\![M]\!] \to [\![M]\!]$$

On the left of $\mapsto$ the rule is shown and on the right both the corresponding extracted constraint term taken from Figure 3–10 and the translation of the rule into a morphism of $\mathbf{DB}_{=}$ taken from Figure 5–1. It is to be checked that

$$id \;=\; [z \vdash z]^*$$

which follows from the definition of identities in $\mathbf{DB}_{=}$.

- The *weak* rule has the translations

$$\frac{\Gamma, M \vdash N}{\Gamma \vdash N} \quad \mapsto \quad \frac{\bar{z}, z^{|M|} \vdash t : |N|}{\bar{z} \vdash t : |N|} \qquad \frac{m \circ \pi_1 : [\![\Gamma]\!] \wedge [\![M]\!] \to [\![N]\!]}{m : [\![\Gamma]\!] \to [\![N]\!]}$$

Suppose $\bar{z} = z_n, \ldots, z_1$ and $n \geq 1$. In this case we must verify the equation

$$m \circ \pi_1 \;=\; [\bar{z}, z \vdash t]^*$$

under the inductional hypothesis that $m = [\bar{z} \vdash t]^*$. Also we know that z has type $|M|$ and is not free in t. Observe that by definition the projection π_1 :

$\llbracket\Gamma\rrbracket \wedge \llbracket M\rrbracket \to \llbracket\Gamma\rrbracket$ in $\mathbf{DB}_=$ is the equivalence class $[w \vdash \pi_1(\pi_2 w)]$ and composition $f \circ g$ in $\mathbf{DB}_=$ is $f \circ \langle \pi_1, g\rangle$ in $\mathcal{T}_=$. Now we compute

$$\begin{aligned}
m \circ \pi_1 &= m \circ \langle \pi_1, [w \vdash \pi_1(\pi_2 w)]\rangle \\
&= m \circ \langle [w \vdash \pi_1 w], [w \vdash \pi_1(\pi_2 w)]\rangle \\
&= m \circ [w \vdash (\pi_1 w, \pi_1(\pi_2 w))] \\
&= [\bar{z} \vdash t]^* \circ [w \vdash (\pi_1 w, \pi_1(\pi_2 w))] \\
&= [w \vdash t\{\pi_n^n(\pi_2 w)/z_n\} \cdots \{\pi_1^n(\pi_2 w)/z_1\}] \circ [w \vdash (\pi_1 w, \pi_1(\pi_2 w))] \\
&= [w \vdash t\{\pi_n^n(\pi_2(\pi_1 w, \pi_1(\pi_2 w)))/z_n\} \cdots \{\pi_1^n(\pi_2(\pi_1 w, \pi_1(\pi_2 w)))/z_1\}] \\
&= [w \vdash t\{\pi_n^n(\pi_1(\pi_2 w))/z_n\} \cdots \{\pi_1^n(\pi_1(\pi_2 w))/z_1\}] \\
&= [w \vdash t\{\pi_{n+1}^{n+1}(\pi_2 w)/z_n\} \cdots \{\pi_2^{n+1}(\pi_2 w)/z_1\}] \\
&= [w \vdash t\{\pi_{n+1}^{n+1}(\pi_2 w)/z_n\} \cdots \{\pi_2^{n+1}(\pi_2 w)/z_1\}\{\pi_1^{n+1}(\pi_2 w)/z\}] \\
&= [z_n, \ldots, z_1, z \vdash t]^*
\end{aligned}$$

Notice that in the fifth line above we are using variable w with two different types: In the left morphism of the composition $\circ$, w has type $() \times |\Gamma|$ while in the right morphism it has type $() \times (|\Gamma| \times |M|)$, where $|\Gamma| = |M_n| \times \cdots \times |M_1|$. Now suppose $n = 0$. In this case the goal is to verify

$$m \circ ! \;=\; [w \vdash t]^*$$

The morphism $! :: \llbracket M\rrbracket \to \mathit{true}$ is $[w \vdash *]$, whence under the induction hypothesis $m = [\vdash t]^*$ one finds:

$$\begin{aligned}
m \circ ! &= m \circ \langle \pi_1, [w \vdash *]\rangle \;=\; m \circ [w \vdash (\pi_1 w, *)] \\
&= [\vdash t]^* \circ [w \vdash (\pi_1 w, *)] \;=\; [w \vdash t] \circ [w \vdash (\pi_1 w, *)] \\
&= [w \vdash t] \;=\; [w \vdash t\{\pi_2 w/z\}] \\
&= [w \vdash t]^*
\end{aligned}$$

since z does not occur free in t.

- The rule *falseE* has the translations

$$\mathit{false} \vdash M \quad \mapsto \quad z^0 \vdash \Box z : |M| \qquad \Box : \mathit{false} \to \llbracket M\rrbracket$$

We have to show that $\Box = [z^{\mathbf{0}} \vdash \Box z]^*$ which is trivial by definition of morphism $\Box$ in $\mathbf{DB}_=$.

- The rule $\Diamond I$ has the translations

$$M \vdash \Diamond M \quad \mapsto \quad z^{|M|} \vdash ([\,], z) : \Omega^* \times |M| \qquad \eta : [\![M]\!] \to \Diamond [\![M]\!]$$

The monad $\Diamond$ on $\mathbf{DB}_=$ is defined so that $\eta = e \times id$ where $e : () \to \Omega^*$ is the void constraint over $()$ for the notion of constraint $(\Omega^*, [\,], @)$. Thus, $e = [\vdash [\,]]$ and $\eta = [w \vdash ([\,], \pi_2 w)] = [z \vdash ([\,], z)]^*$. This was to be shown.

- The rule $\Diamond M$ has the translations

$$\Diamond\Diamond M \vdash \Diamond M \quad \mapsto \quad z^{\Omega^* \times (\Omega^* \times |M|)} \vdash (\pi_1(\pi_2 z) @ \pi_1 z, \pi_2(\pi_2 z)) : \Omega^* \times |M|$$
$$\mu : \Diamond\Diamond [\![M]\!] \to \Diamond [\![M]\!]$$

The natural transformation μ is defined such that

$$\begin{aligned}
\mu &= \langle (\pi_1 \circ \pi_2 \circ \pi_2) \cdot (\pi_1 \circ \pi_2), \pi_2 \circ \pi_2 \circ \pi_2 \rangle \\
&= \langle [w \vdash \pi_1(\pi_2(\pi_2 w))] \cdot [w \vdash \pi_1(\pi_2 w)], [w \vdash \pi_2(\pi_2(\pi_2 w))] \rangle \\
&= \langle [w \vdash \pi_1(\pi_2(\pi_2 w)) @ \pi_1(\pi_2 w)], [w \vdash \pi_2(\pi_2(\pi_2 w))] \rangle \\
&= [w \vdash (\pi_1(\pi_2(\pi_2 w)) @ \pi_1(\pi_2 w),\ \pi_2(\pi_2(\pi_2 w))] \\
&= [z \vdash (\pi_1(\pi_2 z) @ \pi_1 z\ ,\ \pi_2(\pi_2 z))]^*
\end{aligned}$$

The first equation is the definition of μ as in Theorem 5.2.19 and the second evaluates this definition for the concrete category $\mathbf{DB}_=$. The third line instantiates the particular notion of constraint, *cf.* Lemma 5.2.17.

- The rule $\Diamond F$ has the translations

$$\frac{\Gamma, \Diamond M \vdash \Diamond N}{\Gamma, M \vdash N} \quad \mapsto \quad \frac{\dfrac{\bar{z}, z^{\Omega^* \times |M|} \vdash (\pi_1 z, t\{\pi_2 z/v\}) : \Omega^* \times |N|}{\bar{z}, v^{|M|} \vdash t : |N|}}{\dfrac{\Diamond m \circ \sigma : [\![\Gamma]\!] \wedge \Diamond [\![M]\!] \to \Diamond [\![N]\!]}{m : [\![\Gamma]\!] \wedge [\![M]\!] \to [\![N]\!]}}$$

We have to show

$$\Diamond m \circ \sigma \quad = \quad [\bar{z}, z \vdash (\pi_1 z, t\{\pi_2 z/v\})]^*$$

under the induction hypothesis that $m = [\bar{z}, v \vdash t]^*$. Without loss of gnerality, $\bar{z} = z_n, \ldots, z_1$, $n \geq 1$. $\Diamond$ is defined so that $\Diamond m = (\pi_1 \circ \pi_2, m \circ (id \times \pi_2))$, where $\pi_1 \circ \pi_2 = [w \vdash \pi_1(\pi_2 w)]$ and $id \times \pi_2 = [w \vdash (\pi_1 w, \pi_2(\pi_2 w))]$. By induction hypothesis,

$$m = [w \vdash t\{\pi_{n+1}^{n+1}(\pi_2 v)/z_n\} \cdots \{\pi_2^{n+1}(\pi_2 w)/z_1\}\{\pi_1^{n+1}(\pi_2 w)/w\}]$$

Thus,

$$\Diamond m = [w \vdash (\pi_1(\pi_2 w)\,,\ t\{\pi_{n+1}^{n+1}(\pi_2(\pi_2 v))/z_n\} \cdots \{\pi_2^{n+1}(\pi_2(\pi_2 w))/z_1\}\{\pi_1^{n+1}(\pi_2(\pi_2 w))/w\})]$$

Now we compute $\Diamond m \circ \sigma$. The strength σ is the morphism

$$\begin{aligned} \sigma &= \langle \pi_1 \circ \pi_2 \circ \pi_2, \langle \pi_1 \circ \pi_2, \pi_2 \circ \pi_2 \circ \pi_2 \rangle \rangle \\ &= [w \vdash (\pi_1(\pi_2(\pi_2 w)), (\pi_1(\pi_2 w), \pi_2(\pi_2(\pi_2 w))))] \end{aligned}$$

For simplicity let us identify σ with the term on the right of $\vdash$, *i.e.* $\sigma = [w \vdash \sigma]$. Then,

$$\begin{aligned} &\Diamond m \circ \sigma \\ &= \Diamond m \circ \langle \pi_1, \sigma \rangle = \Diamond m \circ [w \vdash (\pi_1 w, \sigma)] \\ &= [w \vdash (\pi_1(\pi_2(\pi_1 w, \sigma))\,, \\ &\qquad t\{\pi_{n+1}^{n+1}(\pi_2(\pi_2(\pi_1 w, \sigma)))/z_n\} \cdots \{\pi_2^{n+1}(\pi_2(\pi_2(\pi_1 w, \sigma)))/z_1\} \\ &\qquad \{\pi_1^{n+1}(\pi_2(\pi_2(\pi_1 w, \sigma)))/v\})] \\ &= [w \vdash (\pi_1 \sigma\,, \\ &\qquad t\{\pi_{n+1}^{n+1}(\pi_2 \sigma)/z_n\} \cdots \{\pi_2^{n+1}(\pi_2 \sigma)/z_1\}\{\pi_1^{n+1}(\pi_2 \sigma)/v\})] \\ &= [w \vdash (\pi_1(\pi_2(\pi_2 w))\,, \\ &\qquad t\{\pi_n^n(\pi_1(\pi_2 \sigma))/z_n\} \cdots \{\pi_1^n(\pi_1(\pi_2 \sigma))/z_1\}\{\pi_2(\pi_2 \sigma)/v\})] \\ &= [w \vdash (\pi_1(\pi_2(\pi_2 w))\,, \\ &\qquad t\{\pi_n^n(\pi_1(\pi_2 w))/z_n\} \cdots \{\pi_1^n(\pi_1(\pi_2 w))/z_1\}\{\pi_2(\pi_2(\pi_2 w))/v\})] \\ &= [w \vdash (\pi_1(\pi_1^{n+1}(\pi_2 w))\,, \end{aligned}$$

$$t\{\pi_{n+1}^{n+1}(\pi_2 w)/z_n\}\cdots\{\pi_2^{n+1}(\pi_2 w)/z_1\}\{\pi_2(\pi_1^{n+1}(\pi_2 w))/v\})]$$

$$= [w \vdash (\pi_1 z\,,\, t\{\pi_2 z/v\})$$
$$\{\pi_{n+1}^{n+1}(\pi_2 w)/z_n\}\cdots\{\pi_2^{n+1}(\pi_2 w)/z_1\}\{\pi_1^{n+1}(\pi_2 w)/z\}]$$

$$= [\bar{z}, z \vdash (\pi_1 z\,,\, t\{\pi_2 z/v\})]^*$$

which was to be demonstrated.

∎

Theorems 5.2.21 and 5.2.23 now imply correctness of the constraint extraction for $\mathcal{L}_0$, more precisely the second part of Lemma 3.2.2 of Section 3.2:

Corollary 5.2.24 *Given a derivation $M_n, \ldots, M_1 \vdash M$ in $\mathcal{L}_0$ and a sequence $z_n, \ldots, z_1$ of fresh variables, $n \geq 1$, of types $|M_i|$, $i = 1, \ldots, n$. Let $z_n, \ldots, z_1 \vdash t : |M|$ be the constraint term constructed as in Sec. 3.2. Then, there is a derivation of the sequent*

$$M_n{}^{\#}\, z_n, \ldots, M_1{}^{\#}\, z_1 \vdash_{z_n,\ldots,z_1} M^{\#}\, t.$$

in the base logic.

Proof: For simplicity let us assume that $n = 1$. The general case can be treated in the same way. The derivation of $M_1 \vdash M$ gives rise to a morphism $m : [\![M_1]\!] \to [\![M]\!]$ such that $m = [z_1 \vdash t]^*$. Theorem 5.2.21 gives us domain and codomain of m:

$$m \;:\; (|M_1|, [z_1 \vdash M_1{}^{\#}\, z_1]^*) \to (|M|, [z \vdash M^{\#}\, z]^*)$$

By construction of morphisms in $\mathbf{DB}_=$ we know that

$$[z_1 \vdash M_1{}^{\#}\, z_1]^* \;\sqsubseteq\; \mathbf{B}_=[\langle \pi_1, [z_1 \vdash t]^* \rangle][z \vdash M^{\#}\, z]^*$$

A little computation shows that the right side is equal to $[z_1 \vdash M^{\#}\, t]^*$, and it is similarly easy to see that $[z_1 \vdash M_1{}^{\#}\, z_1]^* \sqsubseteq [z_1 \vdash M^{\#}\, t]^*$ implies

$$M_1{}^{\#}\, z_1 \vdash_{z_1} M^{\#}\, t$$

by definition of $\sqsubseteq$.

∎

Chapter 6

Related Work

Behavioural constraints abound in hardware engineering. Practically relevant design methodologies based on circuit behaviour have to accommodate constraints in one way or other. Convincing examples may be found in Subrahmanyam's expositions [Sub88a,Sub88b], which give a good idea of how timing abstraction provides a rich source of behavioural constraints. In algebraic modelling of circuit behaviour, such as those based on automata [BC88,LBC88] or those based on processes [Dav88,Tra87], constraints are treated as first-class behaviours and the verification of a constraint is reduced to the comparison of behaviours. Special-purpose theorem provers, such as SILICA PITHECUS [Wei90] for the verification of synchronous MOS circuits or BEAVER [HN89b] for the functional and timing verification of synchronous systems above the gate level, contain sophisticated built-in constraint handling as an essential part of behavioural specification and analysis.

Despite its importance in hardware engineering, however, only recently was the question addressed of formalizing the concept of constraints in modern general-purpose theorem provers currently applied to the formal verification of hardware. It was indicated already in Section 2.3.2 that John Herbert's work [Her88b] using HOL [Gor85,Gor88] can be seen as a step in this direction. Another work in this area that we are aware of is Holger Busch's [Bus91], which investigates

the proof-based transformation of circuit descriptions using the LAMBDA [FM89] theorem prover. Both approaches will be discussed below.

The lack of attention paid to constraints in applying a general-purpose theorem prover to the formal verification of hardware can be explained in several ways. For one, the concept of a constraint has a great variety of facets in practice, making it very difficult to associate a precise meaning with it, let alone to formalize it in terms of mathematical logic. In the context of particular design methodologies few attempts in clarifying the notion have been made [Wei86, Tra87,Sub88a,Dav88,DM88]. Most of these discussions come down to regarding constraints as restrictions which are *deliberately introduced* to simplify the specification or verification task. Hence, the question of how temporarily to brush constraints under the carpet does not pose itself. This contrasts with the stand taken in this thesis, namely to consider constraints as *unwanted by-products* of formalizing abstractions. Another reason why constraints are not perceived to be of interest might be the following: When it comes to formalizing constraints on a theorem prover, constraints are first of all propositions, and as propositions they are part of the specification. In this frame of mind, it is tempting to settle with the conclusion that if one knows how to deal with specifications then one knows how to deal with constraints.

So much for a general introduction to the role of constraints in the context of formal hardware verification. Let us now turn to a more concrete comparison with related work. The main features of our research described in this thesis which distinguish it from most other treatments in the field, so we believe, are the following: We argue that constraints both deserve and require special consideration, in particular that there is good reason to distinguish between constraints and specifications. We model the notion of 'correctness up to constraints' as a modality of predicate logic. We describe a novel method of handling constraints that takes constraints out of the propositions and makes them part of their proofs, with the benefit that constraint manipulation then is induced by computational

semantics for proofs, a concept well-known from mathematical investigations of constructive logics. In the rest of this chapter these three main features, which fall under the aspects *hardware verification*, *modal logics*, and *proof semantics*, will be illuminated by discussing related work.

Hardware Verification

The two relevant approaches of dealing with constraints in the formal verification of hardware are [Bus91] and [Her88b]. To begin with, we note that both approaches are restricted to the modelling of circuit behaviour as predicates while lax logic is applicable not only to the components-as-predicates but also to the functional components-as-functions paradigm.

Holger Busch in [Bus91] uses the general purpose interactive theorem prover LAMBDA to implement a transformation system for behavioural circuit descriptions in which constraints are paid special attention. He takes an algebraic approach based on the elementary notions *component*, *composition*, *inclusion*, and *equivalence* of components. These algebraic 'primitives' are encoded in LAMBDA in a natural way: Components are modelled as predicates or boolean-valued functions which describe an input-output behaviour in a relational way, composition is provided by general higher-order combinators, inclusion is logical entailment or implication, and equivalence is biimplication. Due to the identification of components with arbitrary relations, a constraint, in this framework, can be viewed as a special kind of component. It is called a *pseudo-component* in [Bus91].

For instance, if $A(x,z)$ and $B(z,y)$ are predicates describing two hardware components with x,y,z their signal ports, then their composition is defined as

$$(comp\ A\ B)(x,y) \quad \stackrel{df}{\equiv} \quad \exists z.\ A(x,z) \wedge B(z,y).$$

Let $\gamma(z)$ be a predicate expressing some constraint on signal z. It is transformed into a pseudo-component with ports x,z by

$$\gamma^*(x,z) \quad \stackrel{df}{\equiv} \quad x = z \wedge \gamma(z).$$

Now, in order to impose constraint γ on signal z of component $B(z,y)$ one simply composes B with the pseudo-component γ^*:

$$(comp\,\gamma^*\,B)(x,y) \;=\; \exists z.\, x = z \wedge \gamma(z) \wedge B(z,y) \;\cong\; \gamma(x) \wedge B(x,y).$$

Thus, according to [Bus91] imposing a constraint on signals is reduced to composition of components. This view of constraints has a few immediate consequences that contrast with the approach taken in this thesis: First, constraints in [Bus91] are always constraints on signals. In lax logic constraints can be imposed on arbitrary types, and we have seen that other constraints such as on time (sampling at clock ticks) or on structural parameters (bit-length of incrementor) indeed occur in practice. A second consequence of Busch's approach is that constraints always appear as part of the implementation, *i.e.* the fact that component B satisfies specification A under constraint γ, would be expressed by the sequent $(comp\,\gamma^*\,B)(x,y) \vdash_{x,y} A(x,y)$, or

$$\gamma(x) \wedge B(x,y) \vdash_{x,y} A(x,y)$$

while in lax logic constraints are part of the specification, *i.e.* the verification goal is *let's-not-bother*, $B(x,y) \vdash_{x,y} \Diamond A(x,y)$, or after constraint extraction

$$B(x,y) \vdash_{x,y} \gamma(x) \supset A(x,y).$$

Hence, in [Bus91] constraints *strengthen* the *implementation* while in lax logic they *weaken* the *specification*. We believe that the latter view is more natural since it analyses the offset of an approximate implementation in terms of the intended ideal specification. But of course both views are equivalent.

We note finally that Busch's approach can be viewed essentially as an implementation in LAMBDA of a fragment of Mary Sheeran's RUBY language [JS90] which provides a more general abstract foundation for transformational design based on relations. Similar remarks to those made above apply for comparing [JS90] with our approach in what regards the handling of constraints.

John Herbert in [Her88b] formalizes the temporal abstraction underlying synchronous circuits using the HOL theorem prover. Behaviour is represented by arbitrary HOL predicates and he proposes to express the correctness for a low-level behaviour *wrt.* a high-level behaviour by a statement of the form

$$(\textit{low-level behaviour} \wedge \textit{input stability conditions}) \supset$$
$$(\textit{high-level behaviour} \wedge \textit{output stability assertions})$$

Shaping correctness theorems in this way provides for some separation of concerns as it clearly distinguishes predicates pertaining to constraints from predicates pertaining to behaviour. This pairing of constraint and behaviour, however, leaves both aspects potentially intertwined at the level of proofs: One is stumbling either over input stability conditions or output stability assertions whenever one makes use of a behaviour, which means that constraint manipulation and reasoning in abstract terms have to go together in a single proof. Another shortcoming is that a canonical and systematic manipulation of constraints, although suggested, is neither supported nor enforced by simply rearranging a correctness theorem. This can only be built into a notion of proof as it is done by lax logic.

Modal Logics

The main feature of lax logic is the intuitionistic modal operator $\Diamond$. Now, the formal properties of $\Diamond$ much resemble those of a modality of *possibility*. In particular, the rules $\Diamond I$ and $\Diamond M$ are integral part of the various modal logics of the $S4$-type [Che80], as is the special instance

$$\frac{\Diamond M \vdash_\Delta \Diamond N}{M \vdash_\Delta N}$$

of the *lifting* rule $\Diamond F$. But here lies the most obvious difference to our modal system: Rule $\Diamond F$ in $\mathcal{L}$ is

$$\frac{\Gamma, \Diamond M \vdash_\Delta \Diamond N}{\Gamma, M \vdash_\Delta N}$$

which is rather more powerful in that with it lifting can be applied in an arbitrary context of 'passive' hypotheses Γ. This, as we have seen, has the consequence that

it is no longer possible to apply a classical Kripke semantics for $\Diamond$. Compared to the standard modal systems for $\Diamond$ the strong $\Diamond F$ is a speciality of $\mathcal{L}$ besides the fact that — without $\Diamond$ — it is an intuitionistic logic.

Classical modal logics, where $\Diamond$ and the dual modality of *necessity* $\Box$ are interdefinable, have long been studied, *cf.* [Che80]. There does not seem to be much literature on intuitionistic modalities some of it concentrating on the $\Box$-operator. Publications touching on intuitionistic logics with $\Diamond$ that we are aware of are [Cur57,Pra65,PS86].

Curry in [Cur57,Cur52] very briefly sketches a non-classical modality $\Diamond$ that appears to have the rules $\Diamond I$ and $\Diamond F$. He derives these rules from a proof-theoretic interpretation of $\Diamond$ which is closely related to our reading. Basically, he considers a hierarchy

$$\vdash_1 \subseteq \vdash_2 \subseteq \cdots \subseteq \vdash_n \cdots$$

of stronger and stronger deductive systems and takes $\Diamond M$ to mean that M is 'provable in some stronger system', *i.e.* $\vdash_k \Diamond M$ if $\vdash_l M$ for some $l \geq k$. Curry does not elaborate on this system in much detail, however.

Prawitz in [Pra65] considers an extension of intuitionistic predicate logic by an independent modal operator of possibility that has rule $\Diamond I$ and a weaker version of $\Diamond F$, namely where all hypotheses in Γ must be of form $\neg\Diamond C$ for suitable C. In a classical setting, *i.e.* with classical negation, this system coincides with the well-known system $S4$ [Che80].

Plotkin and Stirling [PS86] present a Kripkean analysis of an intuitionistic propositional logic with two modalities $\Box$ and $\Diamond$. Their system in the presence of the law of the excluded middle is equivalent to the system K [Che80]. This means it encompasses a fairly weak notion of $\Diamond$ which has rule $\Diamond F$ without passive hypotheses Γ and which does not have $\Diamond I$ and $\Diamond M$. They prove a general correspondence theorem which allows to view rules $\Diamond I$ and $\Diamond M$ as semantical conditions on the Kripke frame. However, the restriction on rule $\Diamond F$ in their

framework cannot be lifted as they employ still a classical semantics for $\Diamond$, *i.e.* the set of worlds at which an atomic sentence is true need not be upper closed *wrt.* the frame relation that is used to interpret $\Diamond$.

Proof Semantics

A main technical contribution of this thesis is to define a notion of proof (here: constraint term) and realizability (here: constraint predicate) for a first-order intuitionistic predicate logic with a $\Diamond$ modality and an embedded higher-order base logic treated as proof irrelevant.

For ordinary intuitionistic predicate logics without modalities many notions of constructive proof and realizability are known. Examples are Kleene realizability in Heyting arithmetic and variants thereof, see for instance [TvD88], S. Hayashi's computational logic PX [HN89a], or Ch. P.-Mohring's notion of realizability [PM89] for a version of the Calculus of Constructions. Constraint extraction in lax logic must be seen in this tradition: The constraint predicate $M^* z$ of lax logic is a realizability predicate, often written $z \mathbf{r} M$ or z *realizes* M, and the constraint term is a particular realizer for M obtained from a constructive proof of M.

The formal setting of Ch. P.-Mohring's work is closest to ours in that it defines realizability for a typed logic with the realizers extracted being typed lambda terms (more precisely: F_ω programs). In contrast to this, Kleene realizability uses as realizers partial recursive functions coded as Gödel numbers and in PX the realizers are Lisp programs. So, for the $\Diamond$-free fragment of lax logic the process of constraint extraction may be viewed as a version of [PM89], *viz.* a version which is extensional, first-order, and without explicit distinction between informative and non-informative terms. Let us look at an example. To compare

we have to identify the types

$$\left.\begin{array}{l} \mathit{Data} \\ \mathit{Spec} \\ \mathit{Prop} \end{array}\right\} \text{ with the class of } \left\{\begin{array}{l} \text{object level types,} \\ \text{formulae, and} \\ \text{base propositions} \end{array}\right.$$

in lax logic, respectively. Now, in P.-Mohring's calculus, which is higher-order, the conjunction of two formulae $M : \mathit{Spec}$ and $N : \mathit{Spec}$ would be represented by

$$M \wedge N \equiv \forall C : \mathit{Spec}.\, (M \supset N \supset C) \supset C.$$

Suppose M and N are proper formulae, *i.e.* of type *Spec*, in which case they are called *informative*. Then, the realizer extracted from a proof of $M \wedge N$ according to the translation rules given in [PM89] has type

$$|M \wedge N| \equiv \forall C : \mathit{Data}.\, (|M| \supset |N| \supset C) \supset C.$$

This is the second-order coding of the intensional product of the data types $|M|$ and $|N|$. Similarly, disjunction translates into an intensional sum, *etc.* In lax logic we would get the extensional product $|M| \times |N|$ and extensional sum, *etc.*, which explains the first point made above, *viz.* that constraint extraction is an extensional version of [PM89]. Further, if one of the two conjuncts, say M, is a base proposition, *i.e.* of type *Prop* in [PM89], then it is called *non-informative*. In this case the translation directly gives

$$|M \wedge N| \equiv \forall C : \mathit{Data}.\, (|N| \supset C) \supset C$$

which is an intensional copy of $|N|$, basically. Hence, the translation systematically simplifies redundant parts due to non-informative formulae at the syntactic level, while in lax logic, where M would be of form $M \equiv \iota\phi$ one obtains $|M \wedge N| \equiv \mathbf{1} \times |N|$. This, by extensionality of products and $\mathbf{1}$, is isomorphic but not identical to $|N|$. The distinction between informative and non-informative formulae could be exploited for lax logic too, although it destroys the uniformity of defining the constraint type inductively along the structure of formulae. A

third point that needs mention has to do with the first-order nature of constraint extraction as opposed to the more general higher-order translation of [PM89]. The defining clause

$$|\forall x^{\tau}.\, M| \;\equiv\; \tau \Rightarrow |M|$$

for the constraint type of universal quantification given in Chapter 3.2 would not make sense if τ were the type of all formulae. Namely, in the case where M depends on the variable x, $|M|$ too will depend on variable x. This is because $|M|$ depends on its sub-formulae and x, in case $\tau \equiv \mathit{Spec}$, is a sub-formula. So, $|M|$ would be a dependent type, a notion that we do not have in the object language of lax logic. In [PM89] where one translates into F_{ω}, dependent types are available. The clause for second-order quantification there reads

$$|\forall x : \mathit{Spec}.\, M| \;\equiv\; \forall x : \mathit{Data}.\, |M|.$$

The development of the category theoretic interpretation of lax logic in Section 5.2 bears close relations with the work of James McKinna on deliverables. In his thesis [McK92] McKinna gives a category theoretic analysis of *first-order* and *second-order* deliverables. We noted already that the hyperdoctrine $\mathcal{DB}_{=}$ constructed in Section 5.2 can be seen as an (indexed) category of first-order deliverables with *free variables*. While the motivation for deliverables aims at a *programming language* where programs are annotated with (proof-irrelevant) propositional information regarding their correctness, here, in contrast, $\mathcal{DB}_{=}$ is viewed as a *logic* where proofs have been decorated with constraint information.

Pragmatics aside, the mathematical structure of $\mathcal{DB}_{=}$ is between first-order and second-order deliverables. Due to the free variables it is more expressive than first-order deliverables (explicit input-output relationships can be specified in it [McK92]), but it is weaker than second-order since these variables are object variables in $\mathcal{T}$ and do not range over deliverables themselves. This is reflected by the first-order nature of lax logic: we cannot quantify over formulae, a restriction that is important for our definition of constraint extraction.

Finally, we note that [McK92] employs an intensional approach to preserve the computational meaning of programs, which results in the relevant data structures coming out as *semi*-adjunctions and deliverables as a *semi*-cartesian closed category. In contrast, this thesis being mainly concerned with logic sticks to the traditional extensional notions of cartesian closed category and hyperdoctrine.

Chapter 7

Conclusion

The point of departure for the research reported in this thesis is the insight that behavioural constraints both deserve and require special treatment in formal hardware verification, and that the non-trivial question of how to deal with constraints in an adequate way on interactive theorem provers has not yet been addressed in the literature. In this thesis a solution is proposed that captures 'correctness up to constraints' as a modal operator of intuitionistic logic and uses proof extraction techniques to compute and manipulate constraints. Its main contribution is

- to show that it is possible and advantageous to consider constraints as part of proofs rather than of formulae, and
- to propose a particular formal logic in which constraints are constructed systematically in the course of proving a specification.

By applying these ideas, which are novel and, as we believe, applicable also to software engineering,

- constraint computation arises naturally as semantics on proofs,
- constraints on arbitrary data types can be handled, and

- the way components are modelled is not prejudiced, *i.e.* the calculus is applicable under both the "components-as-functions" and the "components-as-predicates" paradigm.

The first point is explicit in the extraction of constraint terms defined in Section 3.2, which may be seen as a semantical interpretation of proofs in lax logic. This semantics is captured by the categorical model constructed in Section 5.2. The second point is that constraints are given a much more general meaning than usual, where they are taken to be restrictions exclusively on input signals. An abstraction of input signals of type $time \Rightarrow value$ typically is made up from an abstraction on *time* and an abstraction on *value*. So if we restrict constraints to signals we have deprived ourselves of the possibility for explaining constraints on signals from constraints on times and on values. We do not explore this possibility but demonstrate that it is useful in fact to work with constraints on other types as well as on time, and that this can be done in our framework without adding extra complexity. The third point is illustrated in Chapter 4 by the decrementor, incrementor, and factorial for the "components-as-functions" paradigm, and the *modulo*-2 counter for the "components-as-predicates" paradigm.

These are the main pragmatic advantages of our quite general treatment of constraints. The central technical aspects of lax logic can be summarized as follows:

Two-Level Logic Lax logic is a first-order intuitionistic logic that embraces a higher-order base logic. It is a consistent and conservative extension, *i.e.* it does not prove any new things about the base logic (Theorems 3.1.16 and 3.1.18). This is an important result since it means that the new features added, *viz.* $\Diamond$ and constraint extraction, are orthogonal to and do not interfere with the structure of the base logic.

Partial Proof Irrelevance Constraint extraction defined in Section 3.2 identifies all proofs that are performed within the base logic. Thus, the base logic is

proof irrelevant, which is very useful for efficiency of implementation since only proofs in the first-order extension of lax logic need to be stored. Through variation in formulating a specification the user has some control over how much of a proof is in the base logic and how much is in lax logic.

Parametric Base Logic The definition of the base logic in Section 3.1.1 and 3.1.2 only nails down minimal requirements. Thus, lax logic need not be seen as a particular and fixed logic but rather as a method for extending one's favourite predicate logic so as to accommodate constraints and approximate specifications.

Unorthodox $\Diamond$ modality The operator $\Diamond$ enjoys rather strong properties which appear unorthodox for a modality of possibility. The objectionable property is the fact that in rule $\Diamond F$, which serves to put $\Diamond$s around hypotheses and the assertion of a sequent, we allow an arbitrary context of passive hypotheses. As shown in Section 5.1 this precludes a classical interpretation of $\Diamond$. It was seen in Section 4.1.4 that this strong $\Diamond F$ rule, or equivalently the rule $\Diamond F^*$ (*cf.* page 134), was the essential key in verifying the synchronous *modulo*-2 counter.

7.1 Further Research

This thesis suggests three natural directions for further research, relating to the implementation, application, and meta-theory of lax logic.

7.1.1 Implementation

In order to assess its practical importance it will be necessary to implement lax logic on a computer and test it on larger case studies.

An early version of lax logic, reported in [Men91a], has been implemented in the interactive proof editor LEGO [LP92] and simple verification examples such as the *modulo-2* counter have been performed using it. In this prototypical implementation the modal operator $\Diamond$ is encoded using the Σ types [Luo91] and

type universes [HP91] supported by LEGO. These examples are however too simple to test the logic's utility for non-academic verification problems. That implementation with its naive encoding of $\Diamond M$ as $\Sigma\gamma : Prop.\, \gamma \supset M$ is insufficient for two reasons: It is not faithful to the intended interpretation of lax logic as laid down in this thesis, since equivalences like $\Diamond M \vee \Diamond N \cong \Diamond(M \vee N)$, which are valid by constraint extraction, cannot be derived in it. Also, since $\Diamond M$ is no longer a proposition in *Prop* but in $Type_0$ (at least), the encoding of the logic has to take place in the predicative type levels. Hence it makes essential use of LEGO's implicit type inference for universe levels which is an unnecessary type-theoretic complication.

Clearly, much work is left to be done here. LEGO has proven to be a convenient and flexible environment for experimenting with prototype logics and a more adequate implementation of lax logic in LEGO should be sought. Alternatively, an implementation on top of other verification systems should be investigated that are tailored to the needs of hardware design, and provide the necessary infrastructure to run larger examples. In particular, LAMBDA [FM89] and VERITAS [HDL89] seem to be promising candidates. In LAMBDA the possibility of programming complex refinement tactics would permit the automation of large portions of constraint analysis and verification for specific circuit design styles like synchronous or speed-independent circuits. Although LAMBDA does not have an explicit notion of proof, its *flexible* meta-variables could be used to accumulate constraint information.

7.1.2 Application Areas

To prepare for realistic applications a particular area of application needs to be picked and its characteristic constraints investigated. The natural area is synchronous hardware design where on could examine the following classes of timing constraints:

- setup and hold times for input signals

- maximum duration and minimal separation of active clock phases
- lower bounds on sampling times when the circuit is known to have assumed a defined state after power-on time.

In the context of synchronous circuits it would be interesting to consider more than one clock signal and negative delays for reasoning about retiming, which crucially relies on the third type of the constraints above.

As an interesting long-shot application for this research we envisage the integration of the interactive synthesis of speed-independent circuits (SICs) and synchronous circuits (SYCs) within one homogeneous framework employing suitable abstraction functions and corresponding timing constraints. Such a framework, which will crucially benefit from systematic handling of constraints, should be capable of handling a hierarchy of descriptions comprising two-phase SICs, four-phase SICs, SYCs with single clock, and SYCs with multiple clocks.

7.1.3 Meta-Theory

We suggest several possibilities to develop further the meta-theory of lax logic.

Lifting Theorems

Theorem 3.1.17 states that if a formula is provable in lax logic, then its projection into the base logic (*i.e.* all $\Diamond$s and ιs removed) is provable, too. We noted that in general the converse is false. An interesting question with immediate practical relevance is to characterize special cases in which the converse does hold, and to find systematic ways of lifting theorems in the base logic to theorems in lax logic. For instance, we conjecture, provided the base logic does not contain propositions and rules other than those defined in this thesis, that every theorem ϕ of the base logic becomes a theorem in lax logic by prefixing its atomic parts by $\Diamond\iota$; further, if ϕ' is this lifted formula then a proof of ϕ' can be obtained constructively from every proof of ϕ. Whether all these lifted proofs are optimal, or maximal, constraintwise is a separate issue which needs to be investigated.

Ordering on Proofs

The process of extracting constraint information comes down to a notion of explicit proofs for sequents in lax logic. To denote that a term $c : |M|$ is the constraint information extracted from some derivation of $\vdash M$ we may write $\vdash c : M$. The extraction rules then can be translated easily into a correct and complete calculus for deriving instances of this new form of judgement.

Constraint analysis in our framework is proof analysis. It appears natural to introduce an ordering on proofs, written $c \sqsubseteq d : M$, say, expressing that constraint information c is 'stronger' than d. Intuitively, one would expect that $c \sqsubseteq d : M$ holds if there is a proof of $M^\bullet\, c$ from $M^\bullet\, d$. Such an ordering is a natural concept since constraint analysis then amounts to extracting the constraint information d from a derivation of M, replacing it by a stronger constraint c, and proving $c \sqsubseteq d : M$. Also, the ordering measures the extent to which formula M has been proven.

Let us make this more concrete. Consider a proof $f : \iota\phi \supset \Diamond\iota\psi$. By definition, f is a constraint term of type $1 \Rightarrow (\Omega^* \times 1)$ such that $\phi \supset \gamma \supset \psi$ is derivable in the base logic, where $\gamma =_{df} \Pi(\pi_1(f\,*))$ is the hidden constraint constructed by f. Performing constraint analysis on f means replacing γ by some other, perhaps *simpler*, assumption γ'. The condition under which this is possible is that

$$\phi \wedge \gamma' \vdash \gamma \tag{7.1}$$

holds, *i.e.* γ' together with ϕ is stronger than γ. In the extreme case where γ' is to be the weakest possible assumption, namely $\gamma' \equiv \mathit{true}$, this amounts to proving γ from ϕ. Given that the hidden assumption γ is an input constraint of a hardware device this will only be possible if ϕ contains complete information about the environment of the device: showing $\phi \vdash \gamma$ amounts to proving that the environment satisfies the input constraints. The typical case, however, will be that ϕ merely describes parts of a complete circuit in which case only parts

of γ will follow from ϕ while other parts have to be retained in γ'. Formally, if $\gamma' \equiv \sqcap(\pi_1(f'*))$, then (7.1) is equivalent to the condition

$$(\iota\phi \supset \Diamond\iota\psi)^* f \;\vdash\; (\iota\phi \supset \Diamond\iota\psi)^* f'$$

or, if this is taken to define $\sqsubseteq$, the condition

$$f' \sqsubseteq f \;:\; \iota\phi \supset \Diamond\iota\psi.$$

The properties of $\sqsubseteq$, assuming an appropriate definition, should be explored and axiomatized in a correct and, if possible, complete way. One would like to show that if $\vdash c : M$ and c is maximal for $\sqsubseteq$, *i.e.* c is a 'weakest' constraint, then M has been proven properly, *i.e.* there is a proof of M' in the base logic, where M' is obtained from M by dropping all $\Diamond$s and ιs.

Categorical Models

From the syntactical data of lax logic we construct in Section 5.2 a first-order hyperdoctrine with some additional structure for an arbitrary notion of constraint. For the concrete notion of constraint $(\Omega^*, [\,], \mathbb{Q})$ the resulting categorical model was shown to capture constraint extraction. More generally, future research should investigate the class of hyperdoctrines with

- a strong monad in each fibre that is preserved by translations, together with
- a reflective sub-hyperdoctrine (*viz.* the base logic) which is represented by a distinguished object Ω in the base

as the intended semantical characterization of lax logic. It may be asked whether lax logic is complete *wrt.* this class, or what the relationship is between this class and the subclass of hyperdoctrines induced by the syntactic calculus together with a concrete notion of constraint.

Specialization to Other Notions of Constraint

In this thesis we are concerned mainly with a very abstract notion of constraint. The formal treatment, however, is developed independently from the notion of constraint and as indicated briefly at the beginning of Chapter 3 other notions can be considered. In fact, for a particular well-defined application more constraint-handling potential can be built in by specializing to the characteristic constraints of the application. For verifying synchronous circuits a simple idea might be to take as constraints natural numbers, denoting upper and lower bounds, and to interpret the operation $\cdot$ on constraints as *max* or *min*, depending on the type of constraints involved.

Generalization to Higher-Order Logic

Another direction of meta-theoretic research is to attempt to extend lax logic by higher-order quantification. As was noted before in Chapter 6 this requires adding dependent types to the object language.

7.2 Open Problems

There are some technical deficiencies associated with the framework of lax logic as laid out in this thesis, which still need to be tackled.

The Kripke semantics in Section 5.1 was shown to be incomplete for the propositional fragment of lax logic. The question might be answered eventually whether there is a modification which is complete. It seems clear that one would have to consider two 'independent' frame relations $\sqsubseteq_i$ and $\sqsubseteq_c$, where the first is used for intuitionistic implication and the latter for $\Diamond$. Also, it might be necessary to add axioms

$$\Diamond \mathit{false} \supset \mathit{false} \qquad \Diamond(M \vee N) \supset \Diamond M \vee \Diamond N$$

to the logic. These formulae would be valid in the Kripke models (provided truth of $\vee$ is decided locally) and also for constraint extraction but apparently they are

not provable in lax logic. Also, an extension of the Kripke analysis covering full lax logic is missing.

In the categorical semantics of Section 5.2 not all features of lax logic have actually been covered. What is omitted is a categorical account of inductive data types such as natural numbers and lists, and also of equality.

The examples treated in Chapter 4 introduce constraints by referring to the global hypothesis *let's-not-bother*. In the context of this hypothesis arbitrary subgoals of form $\bigcirc_\iota \phi$ can be solved by brute force. In the logic as it stands no measures are taken to control this use in any way, which means essentially that the user can make his job very easy by resorting to *let's-not-bother* early on in a proof. For instance, a verification engineer pressed for time might deal with the decrementor

$$\textit{let's-not-bother} \vdash \forall n.\, \bigcirc_\iota(\textit{succ}(\textit{dec}\, n) = n)$$

by applying $\forall I$ and then using *let's-not-bother* immediately to prove the specification $\bigcirc_\iota(\textit{succ}(\textit{dec}\, n) = n)$, which of course means that he has not done any verification work at all. This is revealed by the extracted constraint term $\lambda n.\, ([\textit{succ}(\textit{dec}\, n) = n], *)$, which shows that the whole proof obligation merely is pushed into the constraint. Notice, the potential for such a thing to happen is indicated already by the hypothesis *let's-not-bother* in the lax proof: If there is no hypothesis *let's-not-bother* there is no problem either.

Thus, so far there is no guarantee that there is an upper bound on the strength of a constraint. The open question is whether it is possible, within a restricted proof environment, to get proper control over the constraints introduced. For synchronous systems and minimal clock period as the constraint, for instance, one might imagine a specialized sub-logic that exploits the fact that for every circuit there is a minimal clock period beyond which it operates correctly.

Bibliography

[And86] P. B. Andrews. *An Introduction to mathematical logic and type theory: To truth through proof.* Academic Press, 1986.

[BC88] C. Berthet and E. Cerny. Verification of asynchronous circuits: Behaviours, constraints, and specifications. In G. Birtwistle and P. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*, pages 385–404. Workshop on Hardware Verification, Kluwer Academic Publishers, 1988.

[BJ83] J. C. Barros and B. W. Johnson. Equivalence of the arbiter, the synchronizer, the latch, and the inertial delay. *IEEE Transactions on Computers*, C-32(7):603–614, July 1983.

[BM92] R. M. Burstall and J. H. McKinna. Deliverables: a categorical approach to program development in constructions. Technical Report ECS-LFCS-92-242, Edinburgh University, Department of Computer Science, 1992.

[Brz76] Y. Brzozowsky. *Digital Networks.* Prentice-Hall, 1976.

[Bus91] H. Busch. Proof-based transformation of formal hardware models. In M. Sheeran and G. Jones, editors, *Proceedings of the Workshop on Designing Correct Circuits.* Springer Verlag, 1991.

[BW90] M. Barr and C. Wells. *Category Theory for computing Science.* Prentice Hall, 1990.

[CH88] Th. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76:95–120, 1988.

[Che80] B. Chellas. *Modal Logic.* Cambridge University Press, 1980.

[Chu40] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Cur52] H. B. Curry. The elimination theorem when modality is present. *Journal of Symbolic Logic*, 17:249–265, 1952.

[Cur57] H. B. Curry. *A Theory of Formal Deducibility*, volume 6 of *Notre Dame Mathematical Lectures.* Notre Dame, Indiana, 1957.

[Dav88] Bruce S. Davie. *A formal hierarchical design and validation methodology for VLSI.* PhD thesis, Edinburgh University, Department of Computer Science, October 1988.

[DM88] B. S. Davie and G. J. Milne. Contextual constraints for design and verification. In G. Birtwistle and P. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*, pages 257–265. Workshop on hardware verification, Kluwer Academic Publishers, 1988.

[Fle80] W. I. Fletcher. *An engineering approach to digital design.* Prentice-Hall, Englewood Cliffs, N.J., 1980.

[FM89] M. Fourman and E. M. Mayger. Formally based system design - Interactive hardware scheduling. In G. Musgrave and U. Lauther, editors, *Proceedings of the IFIP TC 10/WG 10.5 International Conference on VLSI, Munich*, pages 101–112, 1989.

[Fou89] M. P. Fourman. The logic of topoi. In J. Barwise, editor, *Mathematical Logic*, pages 1053–1090. North-Holland, 1989.

[Gor85] M. J. C. Gordon. HOL: A machine oriented formulation of higher order logic. Technical Report 68, University of Cambridge, Computer Laboratory, July 1985.

[Gor88] M. J. C. Gordon. HOL: A proof generating system for higher-order logic. In G. Birtwistle and P. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*, pages 73–128. Workshop on Hardware Verification, Kluwer Academic Publishers, 1988.

[HD86] F. K. Hanna and N. Daeche. Specification and verification using higher order logic: A case study. In G. M. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI design, Proc. of the 1985 Edinburgh conf. on VLSI*, pages 179–213. North-Holland, 1986.

[HDL89] F. K. Hanna, N. Daeche, and M. Longley. VERITAS+:a specification language based on type theory. In *Proc. Conf. on Hardware Specification, Verification and Synthesis, Cornell University*, July 1989.

[Hen50] L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15(2):81–91, June 1950.

[Her88a] J. Herbert. Formal verification of basic memory devices. Technical Report 124, University of Cambridge, Computer Laboratory, February 1988.

[Her88b] John Herbert. Temporal abstraction of digital design. In G. Milne, editor, *The fusion of hardware design and verification*, pages 1–25, University of Strathclyde, Glasgow, Scotland, July 1988. IFIP WG 10.2.

[HN89a] S. Hayashi and H. Nakano. *PX, A Computational logic.* MIT press, 1989.

[HN89b] S. H. Hwang and A. R. Newton. BEAVER: a behavioral formal verifier for VLSI design. In L. Claesen, editor, *Applied formal methods for*

correct VLSI design, pages 605–624. Elsevier Science Publishers, B.V. North Holland, 1989.

[HP91] R. Harper and R. Pollack. Type checking, universe polymorphism, and typical ambiguity in the calculus of constructions. *Theoretical Computer Science*, 1991.

[JS90] G. Jones and M. Sheeran. Circuit design in ruby. In J. Staunstrup, editor, *Formal methods for VLSI design*, pages 13–70. North-Holland, 1990.

[KS74] G. M. Kelly and R. Street. Review of the elements of 2-categories. In G. M. Kelly, editor, *Proc. Sydney Category Theory Seminar*, pages 75–103. Springer, 1974.

[Law69] F. W. Lawvere. Adjointness in foundations. *Dialectica*, 28:281–296, 1969.

[LBC88] M. Langevin, C. Berthet, and E. Cerny. Verification of input constraints for synchronous circuits. In G. Milne, editor, *The fusion of hardware design and verification*, pages 137–155, University of Strathclyde, Glasgow, Scotland, July 1988. IFIP WG 10.2.

[LP92] Z. Luo and R. Pollack. LEGO Proof Development System: User's Manual. LFCS Report ECS-LFCS-92-211, Department of Computer Science, University of Edinburgh, 1992.

[LS86] J. Lambek and P. J. Scott. *Introduction to higher order categorical logic.* Cambridge University Press, 1986.

[Luo91] Z. Luo. A higher-order calculus and theory abstraction. *Information and Computation*, 90(1):107–137, 1991.

[Mar86] L. R. Marino. *Principles of computer design.* Computer Science Press, Rockwell, 1986.

[McK92] J. H. McKinna. *Deliverables: A categorical Approach to Program Development in Type theory.* PhD thesis, University of Edinburgh, 1992.

[Mel88] Thomas F. Melham. Abstraction mechanisms for hardware verification. In G. Birtwistle and P. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*, pages 267–292. Workshop on Hardware Verification, Kluwer Academic Publishers, 1988.

[Men91a] M. Mendler. Constrained proofs: A logic for dealing with behavioural constraints in formal hardware verification. In G. Jones and M. Sheeran, editors, *Proceedings of the Workshop on Designing Correct Circuits*, pages 1–28. Springer Verlag, 1991.

[Men91b] M. Mendler. A first-order logic of designs. Talk given at the Workshop on Categorical Methods in the Semantics of Programming Languages, University of Glasgow, April 11–13 1991.

[Mog89] E. Moggi. Computational lambda-calculus and monads. In *4th IEEE Conf. LICS*, 1989.

[Pit89] A. Pitts. Notes on categorical logic. Lecture Notes, University of Cambridge Computer Laboratory, Lent Term 1989.

[PM89] Ch. Paulin-Mohring. Extracting F_ω programs from proofs in the calculus of construction. In *Proceedings 16th ACM Symposium on POPL*, pages 89–104, 1989.

[Pra65] Dag Prawitz. *Natural Deduction. A Proof Theoretic Study*, volume 3 of *Stockholm Studies in Philosophy.* Almquist & Wiksell, 1965.

[PS86] G. Plotkin and C. Stirling. A framework for intuitionistic modal logics. In *Theoretical aspects of reasoning about knowledge*, pages 399–406, Monterey, 1986.

[See83] R. A. G. Seely. Hyperdoctrines, natural deduction and the beck condition. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 20:505–542, 1983.

[Sub88a] P. A. Subrahmanyam. Contextual constraints, temporal abstraction, and observational equivalence. In G. Milne, editor, *The fusion of hardware design and verification*, pages 156–182, University of Strathclyde, Glasgow, Scotland, July 1988. IFIP WG 10.2.

[Sub88b] P. A. Subrahmanyam. Towards a framework for dealing with system timing in very high level silicon compilers. In P. Subrahmanyam G. Birtwistle, editor, *VLSI Specification, Verification, and Synthesis*, pages 159–215. Workshop on Hardware Verification, Kluwer Academic Publishers, 1988.

[TBG89] A. Tarlecki, R. M. Burstall, and J. A. Goguen. Some fundamental algebraic tools for the semantics of computation. part 3: Indexed categories. Internal Report ECS-LFCS-89-90, Edinburgh University, Department of Computer Science, 1989.

[Tra87] Niklas Traub. *A formal approach to hardware analysis.* PhD thesis, Edinburgh University, Department of Computer Science, March 1987.

[TvD88] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics.* North-Holland, 1988.

[Ung69] S. H. Unger. *Asynchronous sequential switching circuits.* Wiley-Interscience, New York, 1969.

[Wei86] D. W. Weise. *Formal multilevel hierarchical verification of synchronous MOS VLSI.* PhD thesis, Massachusetts Institute of Technology, 1986.

[Wei90] D. Weise. Multilevel verification of MOS circuits. *IEEE Transactions on Computer-Aided Design*, 9(4):341–351, April 1990.