M4M 2007

Intuitionistic Letcc via Labelled Deduction

Jason Reed [1,2] Frank Pfenning[3]

Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Abstract

The well-known embedding of intuitionistic logic into classical modal logic means that intuitionistic logic can be viewed as a calculus of labelled deduction on multiple-conclusion sequents, where the labels are the Kripke worlds of the modal embedding. The corresponding natural deduction system constitutes a type system for programs using control operators such as letcc that capture the current program continuation, which has a modal restriction on the use of such continuations that enforces constructive validity. This allows us to develop a rich dependent type theory incorporating letcc, which is known to be otherwise highly problematic for computational interpretations of classical logic.
Moreover, we give a novel constructive proof for the soundness of this labelled deduction system, whose algorithmic content is a non-deterministic translation of programs that eliminates uses of letcc and is fully compatible with dependent types and therefore with program verification. This proof has been formally verified on the propositional fragment in the Twelf meta-logical framework.

Keywords: modal translation, intuitionistic logic, control operators, dependent types

1 Introduction

Griffin's [8] discovery that control operators arise via a propositions-as-types interpretation from classical propositional logic has spawned a number of interesting and successful investigation into the computational meaning of classical proofs (see, for example, [15,13,21]).

However, there remain flies in the ointment. One is the inescapable fact that beyond certain fragments of arithmetic, classical logic is simply not constructive and any effort to extract computational witnesses from classical proofs can not succeed in general. A related observation is that classical logic is difficult to reconcile with dependent types and leads to degeneracies [9]. Finally, the intuitionistic interpretations of types in terms of their canonical forms (that is, values) no longer applies. As a result, we know very little about how to *reason about* functional programs

[1] This work has been supported by the National Science Foundation under grant CCR-0306313.

[2] Email: jcreed@cs.cmu.edu

[3] Email: fp@cs.cmu.edu

containing control operators except for equational reasoning which is intimately connected to the rules of computation and more tractable [18,13].

In a separate thread of investigation going back to Gödel's interpretation of intuitionistic logic in classical modal logic [7], researchers have proposed *labelled deduction* [6] as a means to restrict classical proofs to be intuitionistically (or modally) sound. Generally, these have been presented in the form of matrix, tableaux, or sequent calculi, but natural deduction systems have also been developed [1].

Our starting point is the observation that labelled deductions remain essentially classical, albeit restricted by the labels to be intuitionistically sound. This means that when we interpret labelled natural deduction proofs as programs they will contain control operators, and these control operators retain their familiar operational meaning. We propose one particular such system, which represents a novel Curry-Howard isomorphism for intuitionistic proofs that accommodates the letcc control operator. Moreover, we incorporate universal and existential quantification in a constructively sound manner, thereby permitting reasoning about programs via first-order dependent types. We are not aware of a similarly dependent system for control operators. We conjecture that an extension to full dependent types is straightforward, given that the equational theory of control operators is relatively well understood.

This does not yet answer the question on how labelled proofs including letcc are related to ordinary intuitionistic proofs, which turned out to be a surprisingly difficult question. Almost all proofs for the soundness of labelled deduction with respect to intuitionistic logic are essentially model-theoretic and not constructive, with the exception of Schmitt and Kreitz's which exhibits a translation between proofs [20]. Their translation, however, starts from a matrix proofs and goes through several complex intermediate sequent calculi. The bulk of the technical material in this paper is devoted to exhibiting an explicit relationship between proofs in the two natural deduction calculi and showing its correctness. The translation appears to be inherently non-deterministic and does not preserve the operational semantics. For example, a function of type $A \to B$ in labelled deduction and its image under translation (of the same type) may carry out very different computations, although both will return elements of the correct type B, even in the presence of first-order dependent types. We interpret this as evidence that adding letcc fundamentally alters the nature of computation, even if we restrict it to be constructively sound.

We have formalized our proof for the propositional fragment in the Twelf [17] meta-logical framework, which exhibited some interesting issues regarding the interaction of parametric reasoning for labels and ordinary variables. [4]

How serious a restriction the label system would be in practice is difficult for us to assess at present. Clearly, the proof of the excluded middle $A \vee \neg A$ is not possible for completely unknown A, but it can be realized in many special cases. Approximately, one might say that the label discipline prohibits 'newer' data from being thrown back to 'older' continuations. This, however, does not convey the whole intuition since, for example, closed data are throwable to any continuation no matter when they were created.

[4] The formal proof can be found on-line at
`http://www.cs.cmu.edu/~jcreed/elf/intletcc.tar.gz`

We do not investigate questions of type inference or the operational semantics in this paper, but it is easy to see that extension by intuitionistic letcc is conservative over the usual call-by-value operational semantics of the λ-calculus. Moreover, it seems clear that T-string unification [14] can by used to efficiently augment the usual type inference with label inference.

The remainder of the paper is organized as follows. Sections 2 and 3 describe the systems of labelled deduction we use, in sequent and natural deduction style, respectively. Section 4 describes the translation from labelled natural deduction to intuitionistic logic that eliminates uses of intuitionistcally valid letcc. Section 5 gives some examples of what programs are and are not possible to write using intuitionistic letcc. Section 6 discusses the formal proof of soundess of our translation. The final sections discuss related work and conclusions.

2 The Labelled Sequent Calculus

The judgment of the labelled sequent calculus takes the form $\Gamma \Rightarrow \Delta$ where Γ, Δ are both lists of labelled propositions $A[p]$. The proposition part is, as usual

$$A, B ::= P(t_1, \ldots, t_n) \mid A \wedge B \mid A \vee B \mid A \supset B \mid \top \mid \bot \mid \forall s.A(s) \mid \exists s.A(s)$$

and the first-order terms that may appear in propositions are of the form

$$t ::= s \mid f(t, \ldots, t)$$

where s is used to denote a variable standing for a first-order term.

Labels (typically written $p, q, r, \ldots$) are strings over letters $a, b, \ldots$, of which we assume we have a countably infinite supply. We write $p \leq q$ when p is a prefix of q, i.e. when q is of the form pr for some string r. These labels intuitively stand for the worlds in a possible-worlds semantics, and we use the words ‘world’ and ‘label’ interchangeably in the sequel. Each letter in a string is like an edge in a path from one Kripke world to another directly accessible from it. The prefix relation $p \leq q$ is then exactly the accessibility relation, expressing that q is accessible from (one may think of it as being ‘one possible future’ of) p.

The sequent rules for the propositional fragment for the judgment are as follows. (A similar setup is due to Waaler [**?**]). We tacitly identify contexts which only differ in the order of their elements (i.e. exchange as an explicit structural rule is not required), and for brevity avoid showing in inference rules that decomposed propositions are also propagated upward in a proof (i.e. contraction as an explicit structural rule is not required).

$$\frac{}{\Gamma, A[p] \Rightarrow A[pq], \Delta}\ \mathit{init}$$

$$\frac{\Gamma, A[pa] \Rightarrow B[pa], \Delta}{\Gamma \Rightarrow A \supset B[p], \Delta}\ {\supset}R^a \qquad \frac{\Gamma \Rightarrow A[pq], \Delta \qquad \Gamma, B[pq] \Rightarrow \Delta}{\Gamma, A \supset B[p] \Rightarrow \Delta}\ {\supset}L$$

$$\frac{\Gamma \Rightarrow A[p], \Delta \qquad \Gamma \Rightarrow B[p], \Delta}{\Gamma \Rightarrow A \wedge B[p], \Delta}\ {\wedge}R \qquad \frac{\Gamma, A[p], B[p] \Rightarrow \Delta}{\Gamma, A \wedge B[p] \Rightarrow \Delta}\ {\wedge}L$$

$$\frac{\Gamma \Rightarrow A[p], B[p], \Delta}{\Gamma \Rightarrow A \vee B[p], \Delta}\vee R \qquad \frac{\Gamma, A[p] \Rightarrow \Delta \qquad \Gamma, B[p] \Rightarrow \Delta}{\Gamma, A \vee B[p] \Rightarrow \Delta}\vee L$$

$$\frac{}{\Gamma \Rightarrow \top[p], \Delta}\top R \qquad \frac{}{\Gamma, \bot[p] \Rightarrow \Delta}\bot L$$

Worthy of particular attention are the init rule and the implication right rule. The first expresses that if we hypothesize A at some world p, and we are able to conclude A at a future world pq, then the sequent is satisfied. This embodies the monotonicity property typical of intuitionistic Kripke models. The implication right rule importantly requires that a is fresh, i.e. does not occur anywhere in the conclusion of the rule. If we read the implication right rule bottom-up, it adds A to the current set of hypotheses, and B to the current set of allowed conclusions, both at the new world pa.

Truly classical proofs would otherwise arise from the interaction between the hypothesis A and other conclusions found in Δ. By affixing the fresh a to the world at which A is hypothesized and B is concluded, the system prevents the hypothesis A from applying Δ, and at the same time allows it to be used in concluding B.

For example, consider the sequent

$$\cdot \vdash (A \vee A \supset \bot)[p]$$

An attempt to prove it,

$$\dfrac{\dfrac{\dfrac{pa \not\leq p}{A[pa] \vdash A[p], \bot[pa]}}{\cdot \vdash A[p], A \supset \bot[p]}}{\cdot \vdash (A \vee (A \supset \bot))[p]}$$

fails because the hypothesis A that is created is at a world pa that is not 'before' the world p at which the conclusion A is.

The sequent rules for the first-order connectives are fairly straightforward, and can be found in the appendix.

3 Labelled Natural Deduction

We now derive a system of natural deduction from the above sequent calculus. The differences between it and ordinary proof terms for intuitionistic logic are the constructs $\mathbf{letcc}\, u\, \mathbf{in}\, M$ and $\mathbf{throw}\, M\, \mathbf{to}\, u$ which permit binding and use of continuations, and the presence of world subscripts on the elimination forms **case**, **abort**, **let**, and on the introduction forms λ, Λ.

Proof terms are given by the grammar

$$\begin{array}{rcl} M, N & ::= & x \mid \langle M_1, M_2 \rangle \mid \pi_i M \mid \langle\rangle \mid \mathbf{abort}_q\, M \mid \mathbf{inj}_i\, M \mid \\ & & (\mathbf{case}_p\, M\, \mathbf{of}\, x_1.M_1 \mid x_2.M_2) \mid \lambda_a x.M \mid M_1\ M_2 \mid \\ & & \mathbf{letcc}\, u\, \mathbf{in}\, M \mid \mathbf{throw}\, M\, \mathbf{to}\, u \mid \Lambda_a s.M \mid M \cdot t \mid \\ & & \mathbf{pack}\langle t, M \rangle \mid \mathbf{let}_q \langle s, x \rangle = M\, \mathbf{in}\, N \end{array}$$

Continuation variables u are a separate syntactic class from proof terms, and there are no other ways to form continuations other than naming **letcc**-bound continuation variables. The subscripts on the binders λ, Λ are in fact binding positions; $\lambda_a x.M$ binds the world-label symbol a (as well as x) in the term M. For all kinds of bound variables (terms s, proof terms x, continuations u, labels a) we tacitly apply variables renaming in order to satisfy side conditions or capture-avoiding substitution according to the usual conventions. The subscript on the elimination forms is a string over bound world-label symbols, and it serves to indicate at which world type-checking of the eliminated-type subexpression takes place. For example, in $\mathbf{case}_q\, M\, \mathbf{of}\, x.M_1 \mid x.M_2$, the object M being case-analzed will have to be well-typed at world q.

The typing judgment for determining well-formedness of a proof term is $\Sigma; \Gamma \vdash M : A[p]$, where Σ is as before a list of first-order term variables, and Γ is here given by

$$\Gamma ::= \cdot \mid \Gamma, x : A[p] \mid \Gamma, u : A[p]$$

i.e. a list of proof-term variables x and continuation variables u, each with associated type A and world p. The typing rules are:

$$\frac{}{\Sigma; \Gamma, x : A[p] \vdash x : A[pq]}$$

$$\frac{\Sigma; \Gamma, x : A[pa] \vdash M : B[pa]}{\Sigma; \Gamma \vdash \lambda_a x.M : A \supset B[p]} \qquad \frac{\Sigma; \Gamma \vdash M_1 : A \supset B[p] \qquad \Sigma; \Gamma \vdash M_2 : A[pq]}{\Sigma; \Gamma \vdash M_1\ M_2 : B[pq]}$$

$$\frac{\Sigma; \Gamma \vdash M_1 : A[p] \qquad \Sigma; \Gamma \vdash M_2 : B[p]}{\Sigma; \Gamma \vdash \langle M_1, M_2 \rangle : A \wedge B[p]} \qquad \frac{\Sigma; \Gamma \vdash M : A_1 \wedge A_2[p]}{\Sigma; \Gamma \vdash \pi_i M : A_i[p]}$$

$$\frac{\Sigma; \Gamma \vdash M : A_i[p]}{\Sigma; \Gamma \vdash \mathbf{inj}_i\, M : A_1 \vee A_2[p]}$$

$$\frac{\Sigma; \Gamma \vdash M : C_1 \vee C_2[q] \qquad \begin{array}{c} \Sigma; \Gamma, x_1 : C_1[q] \vdash M_1 : A[p] \\ \Sigma; \Gamma, x_2 : C_2[q] \vdash M_2 : A[p] \end{array}}{\Sigma; \Gamma \vdash (\mathbf{case}_q\, M\, \mathbf{of}\, x_1.M_1 \mid x_2.M_2) : A[p]}$$

$$\frac{}{\Sigma; \Gamma \vdash \langle\rangle : \top[p]} \qquad \frac{\Sigma; \Gamma \vdash M : \bot[q]}{\Sigma; \Gamma \vdash \mathbf{abort}_q\, M : A[p]}$$

$$\frac{\Sigma, s[pa]; \Gamma \vdash M : A(s)[pa]}{\Sigma; \Gamma \vdash \Lambda_a s.M : \forall s.A(s)[p]} \qquad \frac{\Sigma; \Gamma \vdash M : \forall s.A[p] \qquad \Sigma \vdash t : \mathrm{term}[pq]}{\Sigma; \Gamma \vdash M \cdot t : A(t)[pq]}$$

$$\frac{\Sigma \vdash t : \mathrm{term}[p] \qquad \Sigma; \Gamma \vdash M : A(t)[p]}{\Sigma; \Gamma \vdash \mathbf{pack}\langle t, M\rangle : \exists s.A(s)[p]}$$

$$\frac{\Sigma; \Gamma \vdash M_1 : \exists s.C(s)[q] \qquad \Sigma, s[q]; \Gamma, x : C(s)[q] \vdash M_2 : A[p]}{\Sigma; \Gamma \vdash \mathbf{let}_q \langle s, x\rangle = M_1 \,\mathbf{in}\, M_2 : A[p]}$$

$$\frac{u : C[q] \in \Gamma \qquad \Sigma; \Gamma \vdash M : C[q]}{\Sigma; \Gamma \vdash \mathbf{throw}\, M \,\mathbf{to}\, u : A[p]} \qquad \frac{\Sigma; \Gamma, u : A[p] \vdash M : A[p]}{\Sigma; \Gamma \vdash \mathbf{letcc}\, u \,\mathbf{in}\, M : A[p]}$$

$$\frac{\Sigma \vdash t_i : \mathrm{term}[p]}{\Sigma \vdash f(t_1, \ldots, t_n) : \mathrm{term}[p]} \qquad \frac{s : \mathrm{term}[p] \in \Sigma}{\Sigma \vdash s : \mathrm{term}[pq]}$$

This system has the same properties with respect to variable use and function formation as the sequent calculus had in its init and implication right rules. A variable can be used at any world later than the world at which it was hypothesized, and forming a function entails hypothesizing a new world pa (for fresh a) in the future of the current one p.

The multiple conclusions of the sequent calculus are effectively replaced by **letcc** and **throw** , which make explicit the negotation of which conclusion is actually satisfied. This relationship between classical sequent calculi and control operators is quite standard; it is the interaction of the labels that is interesting for our purposes.

Since we have the goal of showing the soundness of the labelled multi-conclusion sequent calculus in ordinary natural deduction, we claim first that the labelled multi-conclusion sequent calculus is sound in this labelled natural deduction system:

Theorem 3.1 *If* $\Rightarrow A[p]$*, then there is an* M *such that* $\vdash M : A[p]$*.*

Its proof is not difficult, and deferred to the appendix for space reasons.

We can revisit in this setting the example of why a typical classical proof fails. The term

$$M = \mathbf{letcc}\, u \,\mathbf{in}\, \mathbf{inj}_2(\lambda_a x.\mathbf{throw}\,(\mathbf{inj}_1\, x)\,\mathbf{to}\, u)$$

would be, in a type theory for classical logic, a proof of $A \vee (A \supset \bot)$, but trying to typecheck it in the present system fails for the same world mismatch as described in the sequent calculus above:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{pa \not\leq p}{u : (A \vee (A \supset \bot))[p], x : A[pa] \vdash x : A[p]}}{u : (A \vee (A \supset \bot))[p], x : A[pa] \vdash \mathbf{inj}_1\, x : A \vee (A \supset \bot)[p]}}{u : (A \vee (A \supset \bot))[p], x : A[pa] \vdash \mathbf{throw}\,(\mathbf{inj}_1\, x)\,\mathbf{to}\, u : \bot[pa]}}{u : (A \vee (A \supset \bot))[p] \vdash \lambda_a x.\mathbf{throw}\,(\mathbf{inj}_1\, x)\,\mathbf{to}\, u) : (A \supset \bot)[p]}}{u : (A \vee (A \supset \bot))[p] \vdash \mathbf{inj}_2(\lambda_a x.\mathbf{throw}\,(\mathbf{inj}_1\, x)\,\mathbf{to}\, u) : (A \vee (A \supset \bot))[p]}}{\cdot \vdash \mathbf{letcc}\, u \,\mathbf{in}\, \mathbf{inj}_2(\lambda_a x.\mathbf{throw}\,(\mathbf{inj}_1\, x)\,\mathbf{to}\, u) : (A \vee (A \supset \bot))[p]}$$

4 Eliminating Intuitionistic Letcc

To show that the two systems described above are conservative over intuitionistic logic, we show that all uses of **letcc** are inessential. Writing M for a typical ordinary natural deduction proof term (whose grammar is just that of M with all world-subscripts erased and $\mathbf{letcc}, \mathbf{throw}$ removed from the language) and $\Gamma \vdash_{\mathrm{ND}} \mathsf{M} : A$ for the ordinary natural deduction typing judgment (whose typing rules are similarly identical to the labelled natural deduction rules except with all labels erased, and **letcc** and **throw** elided), the goal is to show

If $\vdash M : A[p]$, then there exists an M such that $\vdash_{\mathrm{ND}} \mathsf{M} : A$

We show this by giving an explicit non-deterministic translation that produces M from M.

4.1 Answers

The translation is implemented in terms of an auxiliary data structure of expressions called *answers.* What it amounts to is a strengthening of the induction hypothesis that makes the overall theorem go through.

We will show how to translate every labelled proof term into an answer, and how to translate answers into ordinary proof terms. Answers α are built out of ordinary natural deduction proof terms M by

$$\alpha ::= \mathsf{M}{\downarrow} \mid \mathsf{M} \rhd u \mid (\mathsf{M}?_p x_1.\alpha_1 \mid x_2.\alpha_2) \mid \mathsf{M}!_p \mid (\mathsf{M};_p s.x.\alpha)$$

Answers are something intermediate between labelled and ordinary natural deduction terms. They possess labels in subscripts, and may throw to continuations (since the form $\mathsf{M} \rhd u$ is essentially $\mathbf{throw}\,\mathsf{M}\,\mathbf{to}\,u$; see below) and are typed in a labelled context, but do not themselves feature **letcc**.

$\mathsf{M}{\downarrow}$, pronounced 'M done' marks a proof term which only mentions variables whose world labels precede a certain p. This is enforced in the typing rules for answers through a restriction operation $\Gamma{\downharpoonleft_{\le p}}$ on contexts defined by

$$(\Gamma, x : A[q]){\downharpoonleft_{\le p}} = \begin{cases} \Gamma{\downharpoonleft_{\le p}}, x : A & \text{if } q \le p; \\ \Gamma{\downharpoonleft_{\le p}} & \text{otherwise.} \end{cases}$$

$$\cdot{\downharpoonleft_{\le p}} = \cdot \qquad (\Gamma, u : A[q]){\downharpoonleft_{\le p}} = \Gamma{\downharpoonleft_{\le p}}$$

and analogously for term variable contexts Σ. If Γ is a context for labelled natural deduction, then $\Gamma{\downharpoonleft_{\le p}}$ is a context for ordinary natural deduction, consisting of exactly those proof term (and first-order term) variables that are in the past of p. Moreover, all continuation variables are erased. In this way $\mathsf{M}{\downarrow}$ will be a well-formed answer in context Γ at world p if M is a well-formed proof-term in the context $\Gamma{\downharpoonleft_{\le p}}$.

Each of the remaining answer constructors corresponds to a labelled proof-term constructor, except that they rely on similar world-access limitations. $\mathsf{M} \rhd u$ is morally "$\mathbf{throw}\,\mathsf{M}\,\mathbf{to}\,u$," where M cannot refer to any variables except those at worlds prior to the world of the continuation variable u. The expression $\mathsf{M}?_q x.\alpha_1 \mid x.\alpha_2$ is essentially a $\mathbf{case}_q$, $\mathsf{M}!_q$ represents an $\mathbf{abort}_q$, and $\mathsf{M};_q s.x.\alpha$ is an existential elimination analogous to "$\mathbf{let}_q\langle s, x\rangle = \mathsf{M}\,\mathbf{in}\,\alpha$".

Below are the typing rules for answers. The judgment is most generally $\Sigma; \Gamma \vdash \alpha : A[p]$ for a labelled context of variables and continuation variables Γ. Again, Σ is left implicit except in the rule that explicitly manipulates it.

$$\frac{\Gamma\downharpoonleft_{\leq p} \vdash \mathsf{M} : A}{\Gamma \vdash \mathsf{M}\downarrow : A[p]} \qquad \frac{\Gamma\downharpoonleft_{\leq q} \vdash \mathsf{M} : \bot}{\Gamma \vdash \mathsf{M}!_q : A[p]} \qquad \frac{\Gamma\downharpoonleft_{\leq q} \vdash \mathsf{M} : B \qquad u : B[q] \in \Gamma}{\Gamma \vdash \mathsf{M} \triangleright u : A[p]}$$

$$\frac{\Gamma\downharpoonleft_{\leq q} \vdash \mathsf{M} : C_1 \vee C_2 \qquad \begin{array}{c} \Gamma, x_1 : C_1[q] \vdash \alpha_1 : A[p] \\ \Gamma, x_2 : C_2[q] \vdash \alpha_2 : A[p] \end{array}}{\Gamma \vdash (\mathsf{M}?_q x_1.\alpha_1 \mid x_2.\alpha_2) : A[p]}$$

$$\frac{\Sigma\downharpoonleft_{\leq q}; \Gamma\downharpoonleft_{\leq q} \vdash \mathsf{M} : \exists s.A \qquad \Sigma, s[q]; \Gamma, x : B[q] \vdash \alpha : A[p]}{\Sigma; \Gamma \vdash (\mathsf{M};_q s.x.\alpha) : A[p]}$$

There are a few further refinements of this definition that are required to make precise the invariants of the translation. They are separated answers, pre-answers, and pre-separated answers.

4.1.1 Separated Answers

To define separated answers, fix a world-letter a. It is worth noting that the way we introduce fresh symbols in all the rules above means that every occurrence of a given symbol a has a globally unique prefix p that it occurs after. This means we may freely pass between talking about a and pa without fear that there is some $p' \neq p$ such that $p'a$ also occurs.

Some answers have the property that they contain no mention of worlds later than pa, and in them the set of occurrences of pa in subscripts on ?, !, or ; constructs have only pa subscripts in their subterms. For example,

$$\mathsf{M}?_{pa}x.(N!_p) \mid x.(P\downarrow)$$

is an answer that fails to have this property, because p appears below pa. We call answers which have this property *a-separated*.

Formally the class of *a-separated* answers (written $\alpha^{\overline{a}}$) and the class of *a-pure* answers (written α^a, which have the property that every one of their subscripts is exactly pa) is given by the following grammar:

$$\alpha^{\overline{a}} ::= \alpha^a \mid \mathsf{M} \triangleright u \mid \mathsf{M}!_q \mid (\mathsf{M}?_q x.\alpha_1^{\overline{a}} \mid x.\alpha_2^{\overline{a}}) \mid (\mathsf{M};_q s.x.\alpha^{\overline{a}})$$

$$\alpha^a ::= \mathsf{M}\downarrow \mid \mathsf{M}!_{pa} \mid (\mathsf{M}?_{pa}x.\alpha_1^a \mid x.\alpha_2^a) \mid (\mathsf{M};_{pa} s.x.\alpha^a)$$

where q here stands for any world such that $a \notin q$. Note that these are two parametrized families of grammars, with the parameter being the world letter a. Every a that appears on the right of each ::= cannot be instantiated by any a at all (as one might think it could be according to the usual conventions for free metavariables in grammar rules) but rather only exactly the a appearing on the left. The answer $\mathsf{M} \triangleright u$ that represents throwing to a continuation goes in the production for p-separated answers for technical reasons that can be seen from the proof.

4.1.2 Pre-Answers

The translation of terms to answers requires a recursive pass for each proof-term construct. For this we make use of a notion of *pre-answers*, syntactic expressions whose top-level construct is drawn from the language of proof terms, but which below that have the form of answers. The top-level of the translation takes a term, recursively translates its component terms into answers, and then begins an inner recursion to convert the resulting pre-answer into a *bona fide* answer. Formally, pre-answers are given by

$$\beta ::= \langle \alpha_1, \alpha_2 \rangle \mid \pi_i \alpha \mid \mathbf{inj}_i\, \alpha \mid (\mathbf{case}_q\, \alpha\, \mathbf{of}\, x_1.\alpha_1 \mid x_2.\alpha_2)$$

$$\mid \lambda_a x.\alpha^{\overline{a}} \mid \alpha_1\ \alpha_2 \mid \langle\rangle \mid \mathbf{abort}_q\, \alpha \mid \mathbf{letcc}\, u\, \mathbf{in}\, \alpha \mid \mathbf{throw}\, \alpha\, \mathbf{to}\, u$$
$$\mid \Lambda_a s.\alpha^{\overline{a}} \mid \alpha \cdot t \mid \langle \alpha, t \rangle \mid \mathbf{let}_q \langle s, x \rangle = \alpha_1\, \mathbf{in}\, \alpha_2$$

Typing of pre-answers follows exactly the labelled natural deduction rules, except with every M replaced by α. Note that the binders λ_a, Λ_a have bodies that are a-separated answers. We will return to this point in section 4.2.1.

4.1.3 Pre-Separated Answers

Finally, the algorithm that generates separated answers from answers also has an inner loop that requires definition of *a-pre-separated answers*, written $\beta^{\bar{a}}$. An a-pre-separated answer has a single pa-subscripted construct at the root of its expression tree, and is a a-separated answer below that.

$$\beta^{\overline{a}} ::= (M?_{pa}x.\alpha_1^{\overline{a}} \mid \alpha_2^{\overline{a}}) \mid (M;_{pa}\, s.x.\alpha^{\overline{a}})$$

As a-separated answers, a-pure answers, and a-pre-separated answers are all simply refinements of answers, they are typed by the same answer typing rules.

4.2 Translation

Now we can define the translation itself. It is given by five relations

$\alpha \mapsto \mathsf{M}$	Answer α maps back to term M
$\beta^{\overline{a}} \dashrightarrow^{a}_{\mathsf{s}} \alpha^{\overline{a}}$	Pre-separated-answer $\beta^{\overline{a}}$ evaluates to $\alpha^{\overline{a}}$
$\alpha \hookrightarrow^{a}_{\mathsf{s}} \alpha^{\overline{a}}$	Answer α a-separates to answer $\alpha^{\overline{a}}$
$\beta \dashrightarrow \alpha$	Pre-answer β evaluates to answer α
$M \hookrightarrow \alpha$	Labelled term M translates to answer α

The complete rules for these relations are in Appendix A.3. Each of these relations is total and type-preserving in a sense made explicit in 4.3. If an input object is well-typed for the source of one relation, then there exists at least one output object that it is related to, (generally there are many) and every output object that it is related to is also well-typed. The top-level plan of eliminating the

$\mathbf{letcc}, \mathbf{throw}$ proof term M to produce M by translating through answers is then given schematically by

$$M \hookrightarrow \alpha \mapsto \mathsf{M}$$

A typical rule for $\hookrightarrow$ looks underneath the top-level expression construct of some proof-term M, and recursively appeals to itself to translate each component of the expression into an answer. What it is left with is one term constructor applied to answers in place of terms, that is, a pre-answer. The relation $\dashrightarrow$ serves to perform an inner recursion to turn this pre-answer into an answer. For example, some of the rules for pairs are

$$\frac{M_1 \hookrightarrow \alpha_1 \qquad M_2 \hookrightarrow \alpha_2 \qquad \langle \alpha_1, \alpha_2 \rangle \dashrightarrow \alpha'}{\langle M_1, M_2 \rangle \hookrightarrow \alpha'}$$

$$\frac{}{\langle \alpha, \mathsf{M}!_q \rangle \dashrightarrow \mathsf{M}!_q} \qquad \frac{}{\langle \mathsf{M} \rhd u, \alpha \rangle \dashrightarrow \mathsf{M} \rhd u}$$

Using these (and the rules for abort and throw, see Appendix) we can show that e.g. both

$$\langle \mathbf{throw}\, x\, \mathbf{to}\, u, \mathbf{abort}_q\, y \rangle \hookrightarrow x \rhd u$$

$$\langle \mathbf{throw}\, x\, \mathbf{to}\, u, \mathbf{abort}_q\, y \rangle \hookrightarrow y!_q$$

Here the nondeterminism of the relation $\hookrightarrow$ is evident.

4.2.1 Binders and Separation

The notion of separated answers and the separation relations are used for the binding constructs λ_a, Λ_a. The relations $\hookrightarrow^a_{\mathsf{s}}$ and $\dashrightarrow^a_{\mathsf{s}}$ are used to produce from any answer a separated answer of the same type. The remainder of this section is an explanation of the difficulties of the binding cases, and hints at the role that separated answers play in solving them, although a full explanation is beyond the scope of this extended abstract.

Imagine that the λ clause of the relation $\hookrightarrow$ worked in roughly the same way as every other construct, by first recursively applying $\hookrightarrow$ to the body of the lambda, and letting $\dashrightarrow$ eliminate the lambda thereafter:

$$\frac{M \hookrightarrow \alpha \qquad \lambda_a x.\alpha \dashrightarrow \alpha'}{\lambda_a x.M \hookrightarrow \alpha'}$$

We pose the question, then, of what goes in the ? in $\lambda_a x.\alpha \dashrightarrow ?$. First we would certainly split cases on the various possibilities for the answer α. If α is $\mathsf{M}\!\downarrow$, then $\lambda_a x.(\mathsf{M}\!\downarrow) \dashrightarrow (\lambda x.\mathsf{M})\!\downarrow$ is a reasonable response, for it at least makes $\dashrightarrow$ type-preserving in that case. For if the pre-answer $\lambda_a x.(\mathsf{M}\!\downarrow)$ is well-typed, its typing derivation is of the form of the derivation on the left below:

$$\frac{\dfrac{\Gamma|_{\le pa}, x : A \vdash \mathsf{M} : B}{\Gamma, x : A[pa] \vdash \mathsf{M}\!\downarrow : B[pa]}}{\Gamma \vdash \lambda_a x.(\mathsf{M}\!\downarrow) : A \supset B[p]} \qquad \frac{\dfrac{\Gamma|_{\le p}, x : A \vdash \mathsf{M} : B}{\Gamma|_{\le p} \vdash \lambda x.\mathsf{M} : B}}{\Gamma \vdash (\lambda x.\mathsf{M})\!\downarrow : A \supset B[p]}$$

The respective top lines of these derivations are equivalent, by the freshness side-condition on a, so if we have the derivation on the left, we can form the derivation on the right.

Similarly, if the answer underneath the lambda happens to be $\mathsf{M}!_{pa}$, then we can keep the lambda and change the ! to an **abort**, yielding $\lambda_a x.(\mathsf{M}!_{pa}) \dashrightarrow (\lambda x.\, \mathbf{abort}\, \mathsf{M})\downarrow$. We leave it as an exercise to check that this clause is type-preserving. An apparently similar case is the attempt

$$\lambda_a x.(\mathsf{M}!_q) \dashrightarrow (\lambda x.\, \mathbf{abort}\, \mathsf{M})\downarrow \qquad (*)$$

where $a \notin q$, but in fact this is *not* type-preserving, and so not a successful strategy. Here the derivations we have and need are, respectively

$$\dfrac{\dfrac{(\Gamma, x : A[pa])\downharpoonleft_{\leq q} \vdash \mathsf{M} : \bot}{\Gamma, x : A[pa] \vdash \mathsf{M}!_q : B[pa]}}{\Gamma \vdash \lambda_a x.(\mathsf{M}!_q) : A \supset B[p]} \qquad \dfrac{\dfrac{\Gamma\downharpoonleft_{\leq p}, x : A \vdash \mathsf{M} : \bot}{\Gamma\downharpoonleft_{\leq p} \vdash \lambda x.\, \mathbf{abort}\, \mathsf{M} : B}}{\Gamma \vdash (\lambda x.\, \mathbf{abort}\, \mathsf{M})\downarrow : A \supset B[p]}$$

Here the mismatch between the contexts $(\Gamma, x : A[pa])\downharpoonleft_{\leq q}$ and $\Gamma\downharpoonleft_{\leq p}, x : A$ is irreconcilable. On the left, M may refer to some variables in $\Gamma\downharpoonleft_{\leq q}$ that are no longer available in $\Gamma\downharpoonleft_{\leq pa} = \Gamma\downharpoonleft_{\leq a}$.

Fortunately, the fact that $a \notin q$ lets us infer something else: that $(\Gamma, x : A[pa])\downharpoonleft_{\leq q} = \Gamma\downharpoonleft_{\leq q}$, and therefore that x itself cannot appear anywhere in M. We may therefore let the $!_q$ escape the λ, yielding $\lambda_a x.(\mathsf{M}!_q) \dashrightarrow \mathsf{M}!_q$. It is again easy to check that this is type-preserving.

Summarizing, the correct behavior evaluating a $\lambda_a x.$ when we encountering an answer $\mathsf{M}!_q$ depends on the world subscript q. Either (I) q is pa, (q is the 'new' world introduced by the lambda) and we 'imitate' $!_q$ with $\mathbf{abort}_q$, leaving the λ in place, or (II) q does not contain a, (q is some 'old' world) and therefore M doesn't contain x, and we let $\mathsf{M}!_q$ escape the lambda.

However, $!_q$ is not the only form of answer by far; more generally we encounter a tree structure whose internal nodes are $?_q$s and $;_q$s. Can we make a similar simple split between (I) and (II) in every case? It turns out the answer is no. Consider the problem of filling in the blank in

$$\lambda_a x.(\mathsf{M};_{pa}\, s.y.\mathsf{M}';_q\, s'.y'.(\mathsf{M}''\downarrow)) \dashrightarrow ? \qquad (**)$$

We can neither (I) globally imitate with the term

$$\lambda x.(\mathbf{let}\langle s, y\rangle = \mathsf{M}\,\mathbf{in}\,\mathbf{let}\langle s', y'\rangle = \mathsf{M}'\,\mathbf{in}\,\mathsf{M}'')\downarrow$$

(in which M' risks being ill-typed just as M in the output of $(*)$ does above) nor can we (II) drop the lambda as

$$\mathsf{M};_{pa}\, s.y.\mathsf{M}';_q\, s'.y'.(\mathsf{M}''\downarrow)$$

because then a and x (which may appear in M) are suddenly no longer in scope.

The essential problem is that some world-subscripts in the expression are new, and some are old. Strategy (I) only works when the world is new, and strategy

(II) only works when it is old. The solution to this impasse is selectively applying strategies (I) and (II) to the appropriate parts of the expression. The answer to the example ($**$) is, in particular

$$\mathsf{M}';_q s'.y'.(\lambda x.\,\mathbf{let}\langle s,y\rangle = \mathsf{M}\,\mathbf{in}\,\mathsf{M}'')\!\downarrow$$

where $;_q$ has escaped, and $;_{pa}$ has turned into a **let**.

In the interest of making recursive definition of this mixed strategy feasible, the purpose of the *separation relation* $\hookrightarrow^a_{\mathsf{s}}$ is to take as input an answer α, and output an a-separated answer $\alpha^{\overline{a}}$ of the same type, so that all of the answer constructs that need to escape the lambda are already outermost in $\alpha^{\overline{a}}$. The correct $\hookrightarrow$ rule for lambda works by invoking this relation, separating the answer obtained recursively before invoking $\dashrightarrow$:

$$\frac{M \hookrightarrow \alpha \qquad \alpha \hookrightarrow^a_{\mathsf{s}} \alpha' \qquad \lambda_a x.\alpha' \dashrightarrow \alpha''}{\lambda_a x.M \hookrightarrow \alpha''}$$

The $\dashrightarrow$ rules for λ-abstraction each permute the abstraction when the world labels allow it, for example,

$$\frac{a \notin q \qquad \lambda_a x.\alpha \dashrightarrow \alpha'}{\lambda_a x.(\mathsf{M};_q s.y.\alpha) \dashrightarrow \mathsf{M};_q s.y.\alpha'}$$

or reconstruct the abstraction if the subterm is pure:

$$\frac{\alpha^a \mapsto \mathsf{M}}{\lambda_a x.\alpha^a \dashrightarrow (\lambda_a x.\mathsf{M})\!\downarrow}$$

4.3 Correctness

We can concisely state the typing and totality properties of all of these relations with the aid of the following (Hoare-triple-like) abbreviation.

Definition 4.1 When J_1, J_2 are judgments and $\rightsquigarrow$ is a relation, the notation

$$\{J_1(X)\} \quad X \rightsquigarrow Y \quad \{J_2(Y)\}$$

means that both of the following are true:

- ("Progress") If $J_1(X)$, then there is a Y such that $X \rightsquigarrow Y$.
- ("Preservation") For any Y, if $J_1(X)$ and $X \rightsquigarrow Y$, then $J_2(Y)$.

The following theorem is a summary of the progress and preservation properties of the translation.

Theorem 4.2 *Let Γ' be of the form $\Gamma, x_1 : A_1[pa], \ldots, x_n : A_n[pa]$ for some $a \notin \Gamma$, (and similarly for the associated term context $\Sigma' = \Sigma, s_1[pa], \ldots, s_m[pa]$ where $a \notin \Sigma$) and put $q = pa$. All of the following hold:*

$$\begin{array}{llll}
\{\ \Gamma \vdash \alpha^{a} : A[p]\ \} & \alpha^{a} \mapsto \mathsf{M} & \{\ \Gamma\!\downharpoonleft_{\leq p} \vdash \mathsf{M} : A\ \} \\
\{\ \Gamma' \vdash \beta^{\bar{a}} : A[q]\ \} & \beta^{\bar{a}} \dashrightarrow^{a}_{\mathsf{s}} \alpha^{\bar{a}} & \{\ \Gamma' \vdash \alpha^{\bar{a}} : A[q]\ \} \\
\{\ \Gamma' \vdash \alpha : A[q]\ \} & \alpha \hookrightarrow^{a}_{\mathsf{s}} \alpha^{\bar{a}} & \{\ \Gamma' \vdash \alpha^{\bar{a}} : A[q]\ \} \\
\{\ \Gamma \vdash \beta : A[p]\ \} & \beta \dashrightarrow \alpha & \{\ \Gamma \vdash \alpha : A[p]\ \} \\
\{\ \Gamma \vdash M : A[p]\ \} & M \hookrightarrow \alpha & \{\ \Gamma \vdash \alpha : A[p]\ \}
\end{array}$$

Proof. To see that progress holds, one needs to check to see that to every well-formed proof term (or answer, or pre-answer, etc.) there is a clause in the rules defining each appropriate relation that relates that term (resp. answer, pre-answer) to some output.

The proof of preservation proceeds by straightforward structural induction on the relevant typing derivation. ■ □

The main results we want then follow easily from these.

Corollary 4.3 (Soundness of labelled ND) *If* $\vdash M : A[p]$*, then there exists an* M *such that* $\vdash_{\mathrm{ND}} \mathsf{M} : A$*.*

Proof. Choose α such that $M \hookrightarrow \alpha$. It follows that $\vdash \alpha : A[p]$. Choose a fresh label b, and derive the b-pure answer α^{b} from α by changing every label in α to b. It can easily be seen that, after collapsing all labels to b, we have $\vdash \alpha^{b} : A[b]$. Now choose M such that $\alpha^{b} \mapsto \mathsf{M}$. It follows that $\vdash_{\mathrm{ND}} \mathsf{M} : A$, as required. ■ □

Corollary 4.4 (Soundness of labelled sequents) *If there is a derivation of* $\cdot \Rightarrow A[p]$*, then there is* M *such that* $\vdash_{\mathrm{ND}} \mathsf{M} : A$*.*

Proof. Compose Theorem A.5 and Corollary 4.3 ■ □

5 Examples

Treating the natural deduction system as a programming language, it is possible to encode some but not all of the idioms associated with control operators such as **letcc**.

By the soundness theorems immediately above, it is not possible to write closed programs of type $A \vee \neg A$, $\neg\neg A \supset A$, or $((A \supset B) \supset A) \supset A$. However, given the way the typing rules of the **case** construct work, it is possible to present what amounts to a classical proof of, say, $A \vee \neg A$ for certain particular instantiations of A, for example bool. If bool is taken to mean $\top \vee \top$, and we make the obvious definitions of true, false as injections, then the program

$$\begin{aligned} M = \ & \mathbf{letcc}\, u \,\mathbf{in}\, \mathbf{inj}_2(\lambda_a x.\, \mathbf{case}_a\, x\, \mathbf{of} \\ & \quad y.\mathbf{throw}\, (\mathbf{inj}_1\, \mathrm{false})\, \mathbf{to}\, u \\ & \quad \mid y.\mathbf{throw}\, (\mathbf{inj}_1\, \mathrm{true})\, \mathbf{to}\, u) \end{aligned} \qquad (\dagger)$$

type-checks perfectly well as $\cdot \vdash M : \mathrm{bool} + \neg\mathrm{bool}[p]$ for any p, despite its similarity to the program ???hole

$$M = \mathbf{letcc}\, u\, \mathbf{in}\, \mathbf{inj}_2(\lambda_a x.\mathbf{throw}\,(\mathbf{inj}_1\, x)\,\mathbf{to}\, u)$$

which does not type-check, because $x : \mathrm{bool}[pa], u : (\mathrm{bool} \vee \neg\mathrm{bool})[p] \vdash \mathbf{inj}_1\, x : \mathrm{bool} + \neg\mathrm{bool}[pa]$, while u is in the context only at world p, so the throw cannot be typed due to label mismatch. ???hole The example (†) succeeds because in the **case**, the single bit of information that reveals which branch is taken does not itself have a label. The bound variable in each branch of the **case** does, but these bound variables are not used. Instead, the true or false values are closed and valid at any world, and are therefore suitable to be thrown to the continuation u. It is easy to extend this idea to any finite type, thereby recovering a part of the functionality of Kameyama's type restrictions [10].

This program's behavior under translation differs from any term not using letcc: under the algorithm given in the Appendix, it non-deterministically translates to $\mathbf{inj}_1$ true or $\mathbf{inj}_1$ false. Although we make no formal claim about the operational connection between a program and its translation, intuitively this result can be taken to mean that the classical proof eventually takes one or the other of these values once the captured continuation is invoked with a particular boolean.

If we imagine an extension to the system that features recursive functions, then we can also express typical examples such as short-circuiting list product. In SML-like notation,

```
fun $prod\ L$ = letcc $u$ in let
   fun $pd$ [] = 1
        | $pd\ (0 :: tl)$ = throw 0 to $u$
        | $pd\ (n :: tl) = n * pd\ tl$
in $pd\ L$ end
```

still satisfies the label discipline, and its translation is simply the same program, except without the **letcc** u **in**, and with 0 in place of **throw** 0 **to** u. Although the translated program in this case happens to also be a correct implementation of *prod*, we cannot hope for any such luck in general — we emphasize again that the translation does not guarantee the operational equivalence of its output with its input, only that the type is the same. Any invariants of the program that are not captured by the type may not be preserved. The first example above, for instance, already shows that a value that is a second injection into a sum can be transformed into a first injection.

6 Formalization

A proof very similar to the one above has been formalized in the Twelf meta-logical framework. It differs chiefly in that it does not treat first-order quantifiers, but the obstacles to formalizing the entire argument seem to be only inconveniences and are not fundamental to the proof technique.

Another difference is that the formal proof does not feature a translation from explicitly labelled terms, but rather on unlabelled terms. In the algorithm described here, there are many choice points that do one thing or another based on whether a label has some property or not, but in the formal proof these are rephrased in terms of variable occurrences in unlabelled terms. We believe that approach this amounts to the formal proof performing a kind of ad hoc label inference each time it needs to make a decision that is essentially about labels, and that some part of the formalization could in fact be simplified if it could be made to conform more closely to the proof described in this paper. The difficulty with this approach is that, since the proof is to be constructive, one must find a way of representing *evidence* of negative concepts such as inaccessibility of one world from another, and inequality of worlds. Although it is easy enough to say on paper that questions of accessibility and equality of world-strings are decidable, it was convenient to avoid this in our first pass at a formal proof. We plan to consider such an alternative approach in future work, as we extend the formalization to the first-order case.

7 Related Work

The interpretation of intuitionistic propositional logic in classical modal logic goes back to Gödel [7], who gave a translation into the modal logic S4, and claimed its correctness without proof. One direction, that if an intuitionistic proposition is provable, then so too is its modal translation, is relatively easy to show. He conjectured the other direction. A proof was eventually given for the propositional fragment by Tarski and McKinsey [11], who also provided (and proved correct) two other similar translations. A proof for the full first-order case can be found in Fitting's book [5].

The system of classical modal logic that is the target of this translation, and the translation itself, can be at once captured by a system of labelled deduction [6,22]. However, except for [20], the proofs for the soundness of the labelled system that we are aware of are themselves non-constructive, going through some semantics and associated counter-model or universal model argument. (McKinsey & Tarski's uses the topological semantics, while Fitting uses Kripke semantics.) Schmitt and Kreitz's proof is very complicated and, while ultimately syntactic, seems very far from verifiable by formal means and not directly usable for translation from labelled natural deduction to intuitionistic natural deduction.

The $\lambda\mu$-calculus of Parigot [15,16] is a widely known computational interpretation of full propositional classical logic. It has been used by several researchers as the starting point of trying to account for the uses of control operators that remain constructively valid.

Crolard [3,4] studied a restriction of the $\lambda\mu$-calculus equivalent in power to the logic of constant-domain Kripke models, a logic that agrees with intuitionistic logic on the propositional fragment, but admits the non-intuitionistic principle $\forall s.(A(s) \vee B) \supset (\forall s.A(s)) \vee B$ when $s \notin FV(B)$. Crolard argues that this is still a constructive logic, for it still satisfies the disjunction and existential properties. In Crolard's work there can also be found an elucidation of the connection between the $\lambda\mu$-calculus and other λ-calculi with added control operators [2].

Nakano's work [12] can be seen as similar to ours except applied to the (unlabelled) multiple conclusion formulation of intuitionistic logic. In this presentation, constructivity is maintained by constraining the conclusion the implication introduction rule to have a single proposition on the right side of the sequent. The resulting throw and catch primitives are more restrictive than properly labelled letcc.

Kameyama [10] restricts the *types* of data thrown to captured continuations to only allow datatypes such as booleans and lists (and not functional data) to get around the modal restriction of Nakano's system, but this does not extend soundly to dependent types, by Herbelin's counterexample [9].

Sato [19] proves a similar result to ours, that one language with a control operator can be translated to one without, but it is hard to tell in this case whether the tag variables are eliminable in the same sense as world annotations are in our case. Moreover, his control operator seems weaker than ours, more along the lines of Nakano's, and there is no account of first-order or dependent types.

8 Conclusion

We have presented an intuitionistic restriction of a language with classical proof terms including the control operator letcc via a modal logic of labelled deduction. We showed that, unlike unrestricted control operators, this extension is compatible with first-order universal and existential quantification. We also provided a fully formalized constructive proof for the propositional fragment, relating the labelled deduction system back to intuitionistic deduction via a non-deterministic proof translation that gives constructive content to the classic Gödel-McKinsey-Tarski result.

In future work we plan to consider a fully dependent system which requires some characterization of equational reasoning on programs, perhaps along the lines of Ong's elegant system [13].

References

[1] Basin, D., S. Matthews and L. Viganò, *Natural deduction for non-classical logics*, Studia Logica **60** (1998), pp. 119–160.

[2] Crolard, T., *A confluent lambda-calculus with a catch/throw mechanism*, Journal of Functional Programming **9** (1999), pp. 625–647.

[3] Crolard, T., *A constructive restriction of the $\lambda\mu$-calculus*, Technical Report 2002-02, University of Paris 12, LACL (2002).

[4] Crolard, T., *A formulae-as-types interpretation of subtractive logic*, Journal of Logic and Computation **14** (2004), pp. 529–570.

[5] Fitting, M., "Intuitionistic Logic, Model Theory, and Forcing," North-Holland Publishing Co., 1969.

[6] Gabbay, D. M., "Labelled Deductive Systems," vol. 1 **1**, Clarendon Press, Oxford, England, 1996.

[7] Gödel, K., *Eine interpretation des intuitionistischen aussagenkalküls*, Ergebnisse eines mathematischen Kolloquiums **4** (1933), pp. 39–40.

[8] Griffin, T., *A formulae-as-types notion of control*, in: *Conference Record of the 17th Annual Symposium on Principles of Programming Languages (POPL'90)* (1990), pp. 47–58.

[9] Herbelin, H., *On the degeneracy of sigma-types in presence of computational classical logic.*, in: *TLCA*, 2005, pp. 209–220.

[10] Kameyama, Y., *A new formulation of the catch/throw mechanism*, in: T. Ida, A. Ohori and M. Takeichi, editors, *Proceedings 2nd Fuji Intl. Workshop on Functional and Logic Programming, Shonan Village Center*, 1997, pp. 106–122.
URL citeseer.ist.psu.edu/kameyama97new.html

[11] McKinsey, J. C. C. and A. Tarski, *Some theorems about the sentential calculi of Lewis and Heyting*, The Journal of Symbolic Logic **13** (1948), pp. 1–15.

[12] Nakano, H., *A constructive formalization of the catch and throw mechanism*, in: *Logic in Computer Science*, 1992, pp. 82–89.

[13] Ong, L. and C. Stewart, *A Curry-Howard foundation for functional computation with control*, in: *Conference Record of the 24th Annual Symposium on Principles of Programming Languages (POPL'97)* (1997), pp. 215–227.

[14] Otten, J. and C. Kreitz, *T-string-unification: Unifying prefixes in non-classical proof methods*, in: P. Miglioli, U. Moscato, D. Mundici and M. Ornaghi, editors, *5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods* (1996), pp. 244–260.

[15] Parigot, M., *$\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction*, in: *Logic Programming and Automated Reasoning* (1992), pp. 190–201.

[16] Parigot, M., *Classical proofs as programs*, in: *Computational Logic and Theory* (1993), pp. 263–276.

[17] Pfenning, F. and C. Schürmann, *System description: Twelf — a meta-logical framework for deductive systems*, in: H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)* (1999), pp. 202–206.

[18] Sabry, A., *Note on axiomatizing the semantics of control operators*, Technical Report CIS-TR-96-03, University of Oregon (1996).

[19] Sato, M., *Intuitionistic and classical natural deduction systems with the catch and the throw rules*, Theoretical Computer Science **175** (1997), pp. 75–92.
URL citeseer.ist.psu.edu/sato95intuitionistic.html

[20] Schmitt, S. and C. Kreitz, *On transforming intuitionistic matrix proofs into standard-sequent proofs*, in: P. Baumgartner, R. Hähnle and J. Posegga, editors, *4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods* (1995), pp. 106–121.

[21] Wadler, P., *Call-by-value is dual to call-by-name*, in: *ICFP '03: Proceedings of the eighth ACM SIGPLAN international conference on Functional programming* (2003), pp. 189–201.

[22] Wallen, L. A., "Automated Deduction in Non-Classical Logics," MIT Press, 1990.

A Appendix

A.1 First-Order Labelled Natural Deduction

To give rules for the first-order quantifiers, we add a context Σ of term variables to the judgment, so that it becomes $\Sigma; \Delta \Rightarrow \Gamma$. Σ takes the form of a list of term variables, each labelled by a world, $s_1[p_1], \ldots, s_n[p_n]$. In all the rules above, Σ is simply 'passed along' unmodified in each case, and in the sequel we often elide Σ when its behavior is evident.

Σ is, however, manipulated explicitly in connection with first-order terms. The judgment $\Sigma \vdash t : \mathrm{term}[p]$, which asserts the well-formedness of the term t at the world p in the context of term variables Σ, is defined by

$$\frac{}{\Sigma, s[p] \vdash s : \mathrm{term}[pq]} \qquad \frac{\Sigma \vdash t_1 : \mathrm{term}[p] \cdots \Sigma \vdash t_n : \mathrm{term}[p]}{\Sigma \vdash f(t_1, \ldots, t_n) : \mathrm{term}[p]}$$

This judgment is then used to give sequent rules for the quantifiers, as follows:

$$\frac{\Sigma, s[pa]; \Gamma \Rightarrow A(s)[pa], \Delta}{\Sigma; \Gamma \Rightarrow \forall s.A(s)[p], \Delta}\,\forall R^{s,a}$$

$$\frac{\Sigma \vdash t : \mathrm{term}[pq] \qquad \Sigma; \Gamma \Rightarrow A(t)[pq]}{\Sigma; \Gamma, \forall s.A(s)[p] \Rightarrow \Delta} \forall L$$

$$\frac{\Sigma \vdash t : \mathrm{term}[p] \qquad \Sigma; \Gamma \Rightarrow A(t)[p]}{\Sigma; \Gamma \Rightarrow \exists s.A(s)[p], \Delta} \exists R$$

$$\frac{\Sigma, s[p]; \Gamma \Rightarrow A(s)[p], \Delta}{\Sigma; \Gamma, \exists s.A(s)[p] \Rightarrow \Delta} \exists L^s$$

Note that the right rule for $\forall$ is parametric in both the term variable s and the world-label a, while the left rule for $\exists$ is parametric only in a term variable s. In the other two rules, $A(t)$ stands for the replacement of every free occurrence of s in the predicate $A(s)$ with the term t.

A.2 Soundness of Labelled Sequent Calculus in Labelled Natural Deduction

For every sequent proof, $\Gamma \Rightarrow \Delta$, there will be a proof term that exhibits a contradiction from proof-term variables with types from Γ, and continuation variables with types from Δ. What we mean by 'exhibiting a contradiction' is simply the following:

Definition A.1 Let $\Gamma \vdash M : \#$ (read: 'M is a proof of contradiction') abbreviate the fact that $\Gamma \vdash M : A[p]$ for all A, p.

There are a few basic facts about the natural deduction calculus that are easy to show by straightforward structural induction. The first is again strongly reminiscent of the monotonicity properties from the Kripke semantics.

Lemma A.2 (Label Monotonicity) *Suppose $p \leq q$.*

- *If $\Gamma \vdash M : A[p]$, then $\Gamma \vdash M : A[q]$.*
- *If $\Gamma, u : A[p] \vdash M : C[r]$, then $\Gamma, u : A[q] \vdash M : C[r]$.*
- *If $\Gamma, x : A[q] \vdash M : C[r]$, then $\Gamma, x : A[p] \vdash M : C[r]$.*

In the following substitution lemma, $[M/x]N$ stands for the usual capture-avoiding substitution of M for x in N, and $[\![x.M/u]\!]N$ stands for the replacement of every **throw** M' **to** u in N with $[M'/x]M$. The intuition for the latter is that if M can produce a contradiction for any x that would have been the data thrown to u, then the expression M can serve as a replacement for any throw to u.

Lemma A.3 (Substitution) *Suppose*

- *If $\Gamma, x : A[p] \vdash N : B[q]$ and $\Gamma \vdash M : A[p]$, then $\Gamma \vdash ([M/x]N) : B[q]$.*
- *If $\Gamma, u : A[p] \vdash N : B[q]$ and $\Gamma, x : A[p] \vdash M : \#$, then $\Gamma \vdash [\![x.M/u]\!]N : B[q]$*

We now have the tools to state and prove the soundness of the labelled sequent calculus in labelled natural deduction.

Definition A.4 Γ' is an *assignment of variable names* to Γ, Δ if Γ is of the form $A_1[p_1], \ldots, A_n[p_n]$, Δ is of the form $B_1[q_1], \ldots, B_n[q_n]$, and Γ' is of the form

$$x_1 : A_1[p_1], \ldots, x_1 : A_n[p_n], u_1 : B_1[q_1], \ldots, u_n : B_n[q_n]$$

Theorem A.5 *Suppose Γ' is an assignment of variable names to Γ, Δ. If $\Gamma \Rightarrow \Delta$, then there is an M such that $\Gamma' \vdash M : \#$.*

Proof. By induction on the derivation. For example, in the case of the rule $\supset R$,

$$\frac{\Gamma, A[pa] \Rightarrow B[pa], \Delta}{\Gamma \Rightarrow A \supset B[p], \Delta} \supset R^a$$

the induction hypothesis yields a proof term M satisfying $\Gamma', x : A[pa], u : B[pa] \vdash M : \#$. From this we can derive $\Gamma', v : A \supset B[p] \vdash \textbf{throw}\,(\lambda_a x.\,\textbf{letcc}\,u\,\textbf{in}\,M)\,\textbf{to}\,v : \#$ as required. ■ □

Corollary A.6 *If $\Rightarrow A[p]$, then there is an M such that $\vdash M : A[p]$.*

Proof. By the preceding theorem, let M be such that $u : A[p] \vdash M : \#$. In particular, $u : A[p] \vdash M : A[p]$. It follows that $\vdash \textbf{letcc}\,u\,\textbf{in}\,M : A[p]$. ■ □

A.3 Translation

Below is a complete description of the translation from labelled proof terms M (which may use letcc) to ordinary proof terms M (which do not). For compact presentation of the algorithm, we frequently use C to stand for a syntactic expression with a single hole □ somewhere in it, and $\mathsf{C}[X]$ for the replacement of that hole in C by the expression X. $\boxed{R}$ indicates the beginning of the definition of the relation R.

A.3.1 Mapping Answers to Terms

$$\boxed{\alpha \mapsto \mathsf{M}}$$

$$\frac{}{\mathsf{M}{\downarrow} \mapsto \mathsf{M}}$$

$$\frac{}{\mathsf{M}!_p \mapsto \textbf{abort}\,\mathsf{M}}$$

$$\frac{\alpha_1 \mapsto \mathsf{M}_1 \qquad \alpha_2 \mapsto \mathsf{M}_2}{\mathsf{M}?_p x.\alpha_1 \mid x.\alpha_2 \mapsto \textbf{case}\,\mathsf{M}\,\textbf{of}\,x.\mathsf{M}_1 \mid x.\mathsf{M}_2}$$

$$\frac{\alpha \mapsto N}{(\mathsf{M};_p\, s.x.\alpha) \mapsto \textbf{let}\langle s, x\rangle = \mathsf{M}\,\textbf{in}\,N}$$

A.3.2 Separating Answers

$$\boxed{\beta^{\bar{a}} \dashrightarrow^a_{\mathsf{s}} \alpha^{\overline{a}}}$$

If C is from the list

$$(\mathsf{M}?_{pa}y.\square \mid y.\alpha), (\mathsf{M}?_{pa}y.\alpha \mid y.\square), (\mathsf{M};_{pa}\, s.y.\square)$$

then the following rules apply:

$$\dfrac{a \notin q}{\mathsf{C}[\mathsf{M}!_q] \dashrightarrow^{a}_{\mathsf{s}} \mathsf{M}!_q} \qquad \dfrac{}{\mathsf{C}[\mathsf{M} \rhd u] \dashrightarrow^{a}_{\mathsf{s}} \mathsf{M} \rhd u}$$

$$\dfrac{a \notin q \qquad \mathsf{C}[\alpha'_1] \dashrightarrow^{a}_{\mathsf{s}} \alpha''_1 \qquad \mathsf{C}[\alpha'_2] \dashrightarrow^{a}_{\mathsf{s}} \alpha''_2}{\mathsf{C}[\mathsf{M}?_q z.\alpha'_1 \mid z.\alpha'_2] \dashrightarrow^{a}_{\mathsf{s}} \mathsf{M}?_q z.\alpha''_1 \mid z.\alpha''_2}$$

$$\dfrac{a \notin q \qquad \mathsf{C}[\alpha'] \dashrightarrow^{a}_{\mathsf{s}} \alpha''}{\mathsf{C}[\mathsf{M};_q s.z.\alpha'] \dashrightarrow^{a}_{\mathsf{s}} \mathsf{M};_q s.z.\alpha''}$$

The remaining rules defining $\dashrightarrow^{a}_{\mathsf{s}}$ are

$$\dfrac{}{(\mathsf{M}?_{pa} y.\alpha^a_1 \mid y.\alpha^a_2) \dashrightarrow^{a}_{\mathsf{s}} (\mathsf{M}?_{pa} y.\alpha^a_1 \mid y.\alpha^a_2)}$$

$$\dfrac{}{(\mathsf{M};_{pa} s.y.\alpha^a) \dashrightarrow^{a}_{\mathsf{s}} (\mathsf{M};_{pa} s.y.\alpha^a)}$$

$$\boxed{\alpha \hookrightarrow^{a}_{\mathsf{s}} \alpha^{\overline{a}}}$$

The rules defining $\hookrightarrow^{a}_{\mathsf{s}}$ are

$$\dfrac{}{\mathsf{M}{\downarrow} \hookrightarrow^{a}_{\mathsf{s}} \mathsf{M}{\downarrow}} \qquad \dfrac{}{\mathsf{M}! \hookrightarrow^{a}_{\mathsf{s}} \mathsf{M}!} \qquad \dfrac{}{\mathsf{M} \rhd u \hookrightarrow^{a}_{\mathsf{s}} \mathsf{M} \rhd u}$$

$$\dfrac{a \notin q \qquad \alpha_1 \hookrightarrow^{a}_{\mathsf{s}} \alpha'_1 \qquad \alpha_2 \hookrightarrow^{a}_{\mathsf{s}} \alpha'_2}{\mathsf{M}?_q x.\alpha_1 \mid x.\alpha_2 \hookrightarrow^{a}_{\mathsf{s}} \mathsf{M}?_q x.\alpha'_1 \mid x.\alpha'_2}$$

$$\dfrac{\alpha_1 \hookrightarrow^{a}_{\mathsf{s}} \alpha'_1 \qquad \alpha_2 \hookrightarrow^{a}_{\mathsf{s}} \alpha'_2 \qquad \mathsf{M}?_{pa} x.\alpha'_1 \mid x.\alpha'_2 \dashrightarrow^{a}_{\mathsf{s}} \alpha}{\mathsf{M}?_{pa} x.\alpha'_1 \mid x.\alpha'_2 \hookrightarrow^{a}_{\mathsf{s}} \alpha}$$

$$\dfrac{a \notin q \qquad \alpha \hookrightarrow^{a}_{\mathsf{s}} \alpha'}{\mathsf{M};_q x.\alpha \hookrightarrow^{a}_{\mathsf{s}} \mathsf{M};_q x.\alpha'}$$

$$\dfrac{\alpha \hookrightarrow^{a}_{\mathsf{s}} \alpha' \qquad \mathsf{M};_{pa} x.\alpha \dashrightarrow^{a}_{\mathsf{s}} \alpha'}{\mathsf{M};_{pa} x.\alpha \hookrightarrow^{a}_{\mathsf{s}} \alpha'}$$

A.3.3 Evaluating Pre-Answers

$$\boxed{\beta \dashrightarrow \alpha}$$

If C is from the following list:

$$\alpha\,\square, \square\,\alpha, \langle \alpha, \square \rangle, \langle \square, \alpha \rangle, \pi_i \square, \mathbf{inj}_i\,\square,$$

$$\mathbf{case}_q\,\square\,\mathbf{of}\,x.\alpha_1 \mid x.\alpha_2, \mathbf{abort}_q\,\square, \square \cdot t, \langle t, \square \rangle$$
$$\mathbf{let}\langle s, x \rangle = \square\,\mathbf{in}\,\alpha, \mathbf{throw}\,\square\,\mathbf{to}\,u$$

then the following rules apply:

$$\dfrac{}{\mathsf{C}[\mathsf{M}!_q] \dashrightarrow \mathsf{M}!_q} \qquad \dfrac{}{\mathsf{C}[\mathsf{M} \rhd u] \dashrightarrow \mathsf{M} \rhd u}$$

$$\frac{\mathsf{C}[\alpha_1] \dashrightarrow \alpha_1' \qquad \mathsf{C}[\alpha_2] \dashrightarrow \alpha_2'}{\mathsf{C}[\mathsf{M}?_q x.\alpha_1 \mid x.\alpha_2] \dashrightarrow \mathsf{M}?_q x.\alpha_1' \mid x.\alpha_2'}$$

$$\frac{\mathsf{C}[\alpha] \dashrightarrow \alpha'}{\mathsf{C}[\mathsf{M};_q s.x.\alpha] \dashrightarrow \mathsf{M};_q s.x.\alpha'}$$

If C is from the list $\pi_i \Box, \mathbf{inj}_i \Box, \Box \cdot t, \langle t, \Box \rangle$ then the following rule applies:

$$\frac{}{\mathsf{C}[\mathsf{M}\downarrow] \dashrightarrow (\mathsf{C}[\mathsf{M}])\downarrow}$$

The remaining rules defining $\dashrightarrow$ are

$$\frac{}{\mathsf{M}_1\downarrow\ \mathsf{M}_2\downarrow \dashrightarrow (\mathsf{M}_1\ \mathsf{M}_2)\downarrow} \qquad \frac{}{\langle \mathsf{M}_1\downarrow, \mathsf{M}_2\downarrow \rangle \dashrightarrow \langle \mathsf{M}_1, \mathsf{M}_2 \rangle\downarrow}$$

$$\frac{}{\mathbf{throw}\, \mathsf{M}\downarrow \mathbf{to}\, u \dashrightarrow \mathsf{M} \triangleright u} \qquad \frac{}{\mathbf{abort}_q(\mathsf{M}\downarrow) \dashrightarrow \mathsf{M}!_q}$$

$$\frac{}{\mathbf{case}_q(\mathsf{M}\downarrow)\, \mathbf{of}\, x.\alpha_1 \mid x.\alpha_2 \dashrightarrow \mathsf{M}?_q x.\alpha_1 \mid x.\alpha_2}$$

$$\frac{}{\mathbf{let}_q\langle s, x \rangle = (\mathsf{M}\downarrow)\, \mathbf{in}\, \alpha \dashrightarrow \mathsf{M};_q s.x.\alpha}$$

$$\frac{a \notin q}{\lambda_a x.(\mathsf{M}!_q) \dashrightarrow \mathsf{M}!_q} \qquad \frac{}{\lambda_a x.(\mathsf{M} \triangleright u) \dashrightarrow \mathsf{M} \triangleright u}$$

$$\frac{a \notin q \qquad \lambda_a x.\alpha_1 \dashrightarrow \alpha_1' \qquad \lambda_a x.\alpha_2 \dashrightarrow \alpha_2'}{\lambda_a x.(\mathsf{M}?_q y.\alpha_1 \mid y.\alpha_2) \dashrightarrow \mathsf{M}?_q y.\alpha_1' \mid y.\alpha_2'}$$

$$\frac{a \notin q \qquad \lambda_a x.\alpha \dashrightarrow \alpha'}{\lambda_a x.(\mathsf{M};_q s.y.\alpha) \dashrightarrow \mathsf{M};_q s.y.\alpha'}$$

$$\frac{\alpha^a \mapsto \mathsf{M}}{\lambda_a x.\alpha^a \dashrightarrow (\lambda_a x.\mathsf{M})\downarrow}$$

$$\frac{a \notin q}{\lambda_a s.(\mathsf{M}!_q) \dashrightarrow \mathsf{M}!_q} \qquad \frac{}{\lambda_a s.(\mathsf{M} \triangleright u) \dashrightarrow \mathsf{M} \triangleright u}$$

$$\frac{a \notin q \qquad \lambda_a s.\alpha_1 \dashrightarrow \alpha_1' \qquad \lambda_a s.\alpha_2 \dashrightarrow \alpha_2'}{\lambda_a s.(\mathsf{M}?_q y.\alpha_1 \mid y.\alpha_2) \dashrightarrow \mathsf{M}?_q y.\alpha_1' \mid y.\alpha_2'}$$

$$\frac{a \notin q \qquad \lambda_a s.\alpha \dashrightarrow \alpha'}{\lambda_a s.(\mathsf{M};_q s'.y.\alpha) \dashrightarrow \mathsf{M};_q s'.y.\alpha'}$$

$$\frac{\alpha^p \mapsto \mathsf{M}}{\lambda_a s.\alpha^p \dashrightarrow (\lambda_a s.\mathsf{M})\downarrow}$$

$$\frac{u \neq v}{\mathbf{letcc}\, u\, \mathbf{in}\, \mathsf{M} \triangleright v \dashrightarrow \mathsf{M} \triangleright v} \qquad \frac{}{\mathbf{letcc}\, u\, \mathbf{in}\, \mathsf{M} \triangleright u \dashrightarrow \mathsf{M}\downarrow}$$

$$\frac{}{\mathbf{letcc}\, u\, \mathbf{in}\, \mathsf{M}!_q \dashrightarrow \mathsf{M}!_q}$$

$$\frac{\mathbf{letcc}\,u\,\mathbf{in}\,\alpha_1 \dashrightarrow \alpha_1' \qquad \mathbf{letcc}\,u\,\mathbf{in}\,\alpha_2 \dashrightarrow \alpha_2'}{\mathbf{letcc}\,u\,\mathbf{in}(\mathsf{M?}_q x.\alpha_1 \mid x.\alpha_2) \dashrightarrow (\mathsf{M?}_q x.\alpha_1' \mid x.\alpha_2')}$$

$$\frac{\mathbf{letcc}\,u\,\mathbf{in}\,\alpha \dashrightarrow \alpha'}{\mathbf{letcc}\,u\,\mathbf{in}(\mathsf{M;}_q\, s.x.\alpha) \dashrightarrow (\mathsf{M;}_q\, s.x.\alpha')}$$

$$\frac{}{\mathbf{letcc}\,u\,\mathbf{in}\,\mathsf{M}{\downarrow} \dashrightarrow \mathsf{M}{\downarrow}}$$

A.3.4 Translating Terms to Answers

$$\boxed{M \hookrightarrow \alpha}$$

If C is from the following list:

$$\pi_i\Box, \mathbf{inj}_i\,\Box, \mathbf{abort}_q\,\Box, \Box \cdot t, \langle t, \Box\rangle,$$

$$\mathbf{let}\langle s, x\rangle = \Box\,\mathbf{in}\,\alpha, \mathbf{throw}\,\Box\,\mathbf{to}\,u, \mathbf{letcc}\,u\,\mathbf{in}\,\Box$$

then the following rule applies:

$$\frac{M \hookrightarrow \alpha \qquad \mathsf{C}[\alpha] \dashrightarrow \alpha'}{\mathsf{C}[M] \hookrightarrow \alpha'}$$

The remaining rules defining $\hookrightarrow$ are

$$\frac{}{x \hookrightarrow x{\downarrow}}$$

$$\frac{M_1 \hookrightarrow \alpha_1 \qquad M_2 \hookrightarrow \alpha_2 \qquad \langle \alpha_1, \alpha_2\rangle \dashrightarrow \alpha'}{\langle M_1, M_2\rangle \hookrightarrow \alpha'}$$

$$\frac{}{\langle\rangle \hookrightarrow \langle\rangle{\downarrow}}$$

$$\frac{M_i \hookrightarrow \alpha_i (\forall i \in \{0, 1, 2\}) \quad \mathbf{case}_q\,\alpha_0\,\mathbf{of}\,x_1.\alpha_1 \mid x_2.\alpha_2 \dashrightarrow \alpha'}{\mathbf{case}_q\,M_0\,\mathbf{of}\,x_1.M_1 \mid x_2.M_2 \hookrightarrow \alpha'}$$

$$\frac{M \hookrightarrow \alpha \qquad \alpha \hookrightarrow^{a}_{\mathsf{s}} \alpha' \qquad \lambda_a x.\alpha' \dashrightarrow \alpha''}{\lambda_a x.M \hookrightarrow \alpha''}$$

$$\frac{M_1 \hookrightarrow \alpha_1 \qquad M_2 \hookrightarrow \alpha_2 \qquad \alpha_1\ \alpha_2 \dashrightarrow \alpha'}{M_1\ M_2 \hookrightarrow \alpha'}$$

$$\frac{M \hookrightarrow \alpha \qquad \alpha \hookrightarrow^{a}_{\mathsf{s}} \alpha' \qquad \lambda_a s.\alpha' \dashrightarrow \alpha''}{\Lambda_a s.M \hookrightarrow \alpha''}$$