Nominal LCF: A Language for Generic Proof

Jon Sterling

April 27, 2016

Abstract

The syntax and semantics of user-supplied hypothesis names in tactic languages is a thorny problem, because the binding structure of a proof is a function of the goal at which a tactic script is executed. We contribute a new language to deal with the dynamic and interactive character of names in tactic scripts called Nominal LCF, and endow it with a denotational semantics in dl-domains. A large fragment of Nominal LCF has already been implemented and used to great effect in the new RedPRL proof assistant.

1 Introduction

In modern LCF-family proof assistants such as Coq [8], Nuprl [2], JonPRL [15] and Isabelle [10], the user proves theorems by writing programs (“tactics”) in a metalanguage in order to discharge obligations by refinement. A refinement proof system for a logic with hypothetical judgment is usually designed around a sequent calculus, where the goals are essentially of the shape \( H \vdash P \text{ true} \), where \( H \) is a context of named hypotheses \( x : A, y : B, ... \) and \( P \) is a logical proposition.

The LCF methodology is to code the rules of the logic as tactics, namely functions that define the behavior of a rule on a proof state (which typically contains a collection of subgoals as well as a partial proof of the main goal). Tactics can be composed into larger programs, which capture the notion of a derivable rule [13, 11].

As an example of this approach, here is the elimination rule for binary products encoded as a tactic:

\[
\text{elim}_x(z, z_1, z_2)([H, z : A \times B, H' \vdash P \text{ true}]) \rightarrow \\
[H, z : A \times B, z_1 : A, z_2 : B, H', H' \vdash P \text{ true}], \\
[J] \rightarrow [\text{case } z \text{ of } \langle x, y \rangle \Rightarrow [x/z_1, y/z_2]J]
\]

The elimination rule takes three parameters: the hypothesis to target \( z \), and the two fresh names \( (z_1, z_2) \) to use for the generated hypotheses. When the tactic is applied to a goal such that the target hypothesis is a binary product, then the proper subgoal is produced along with a validation which takes the (hypothetical)
evidence of the subgoal and converts it into constructive evidence for the main goal.

This tactic could be used in a script like the following, which splits a product and supplies the left conjunct as the proof of the main goal, assuming that \( z, a, b \) are names in scope:

\[
\text{elim}_{\times}[z,a,b]; \text{hyp}[a]
\]

This design has been used successfully in numerous proof assistants, but there is something inelegant about supplying the names for the generated hypotheses in advance as parameters to the rule. Part of the problem is that in an actual proof, we would intend for the variables \( z_1, z_2 \) to be \textit{bound}, rather than free in the script; in fact, in a typical metalanguage, variables are represented with strings or integers, and so the binding structure of the proof is not at all reflected in the binding structure of the tactic script. It is up to the user to ensure that \( z_1, z_2 \) are fresh, and it is also possible to write a tactic script where hypotheses are matched by coincidence through a name-collision.

It would be desirable to be able to write tactic scripts at a higher level of abstraction, where freshness of names is handled using the binding structure of the ambient metalanguage; that is, it would be preferable if our script could be written in the following way, such that \( a, b \) are bound as hypotheses after the call to the elimination rule:

\[
a, b \leftarrow \text{elim}_{\times}[z]; \text{hyp}[a]
\]

However, there are some obstacles that prevent doing this in a naïve way with existing metalanguages:

1. Hypotheses as they appear in a tactic script are \textit{not} variables: that is, they are characterized by injective/apartness-preserving renaming, not by substitution.

2. In the presence of complex tacticals, the binding location of a supplied name may differ depending on the goal against which the script is run. The “actual” binding structure of the proof synthesized by the evaluated script is essentially a dynamic property rather than a static one.

The second obstacle is perhaps the most devastating one, but we intend to address them both by introducing in Section 3 a new language, \textit{Nominal LCF}, which provides the syntactic structure to capture exactly the dynamic binding of hypotheses that we require, as well as their \textit{nominal} rather than \textit{structural} character.

In Section 4 we give a comparison of \textit{Nominal LCF} with standard LCF. Finally, in Section 5, we give a denotational semantics to \textit{Nominal LCF} inside any LCF-like metalanguage by exploiting a continuity theorem for tactics.
fun MOD F a S =
  let
  val probe = ref 0
  fun β i =
    (probe := Int.max (! probe, i + 1);
     a i)
  val _ = F β S
  in
  !probe
  end

Figure 1: The modulus of continuity for a multitactic implemented in Standard ML using mutable state.

2 Practical Implementation in RedPRL

RedPRL\footnote{RedPRL is the successor to the JonPRL proof assistant \cite{15}, which was originally developed in 2015 by the author and his collaborators (Danny Gratzer and Vincent Rahli) as a standalone implementation of Nuprl’s Constructive Type Theory \cite{2}.} is the successor to the JonPRL proof assistant \cite{15}, which was originally developed in 2015 by the author and his collaborators (Danny Gratzer and Vincent Rahli) as a standalone implementation of Nuprl’s Constructive Type Theory \cite{2}.

One of the most significant improvements of RedPRL over JonPRL and Nuprl lies in its adoption of Nominal LCF, a large fragment of which we have implemented in Standard ML as a highly modular library of signatures and functors.\footnote{The source code for our Nominal LCF implementation is available at \url{https://github.com/JonPRL/sml-dependent-lcf}.} The implementation is essentially an elaboration procedure which follows the denotational semantics given in Section 5 very closely, approximating the domain-theoretic apparatus using Standard ML types.

Following a technique outlined by Longley \cite{7}, we have implemented the modulus of continuity functional defined in Section 5.3 for nominal multitactics by means of Standard ML’s mutable references (Figure 1).

3 Syntax of Nominal LCF

We begin by defining the syntactic sorts of the Nominal LCF theory:

\[
\begin{array}{cccc}
tac sort & mtac sort & tele sort & sym sort \\
\tau_1 sort & \tau_2 sort & \tau_1 \Rightarrow \tau_2 sort
\end{array}
\]

We will need two kinds of contexts: nominal contexts $\Upsilon$ to keep track of nominal...
atoms \(a, b, c\), and variable contexts \(\Gamma\) to keep track of variables \(x, y, z\). They are formed as follows:

\[
\begin{array}{c}
\frac{\text{nomctx}}{Y \text{ nomctx}} \quad a \in Y \\
\frac{\text{varctx}}{\text{varctx}} \quad \tau \text{ sort} \quad x \notin \Gamma \\
\frac{\text{nomctx}}{Y, a \text{ nomctx}} \quad \Gamma, x : \tau \text{ varctx}
\end{array}
\]

The two forms of context differ primarily in the notion of renaming that they support. Whereas a renaming of variable contexts \(\sigma : \Gamma \Rightarrow \Delta\) merely maps every variable in \(\Gamma\) to a variable in \(\Delta\) of the same sort, a renaming of nominal contexts \(\phi : Y \leftrightarrow Y'\) must do so injectively, which reflects the fact that all generated hypotheses are fresh. Moreover, we do not currently assign sorts to nominal atoms in this development.

A signature consists in a family \(\Sigma\) of sets of primitive constants \(c\) or rules, indexed by sorts \(\tau\) and nominal contexts \(Y\) such that renamings of the latter are respected. To be precise, for a constant \(c \in \Sigma(Y, \tau)\) and a renaming \(\phi : Y \leftrightarrow Y'\), we have some \(c \cdot \phi \in \Sigma(Y', \tau)\), such that the renaming action respects identity and composition.

Rather than defining a signature \(\Sigma(Y, \tau)\) functionally, in the future we will informally assert the judgment \(\Sigma \vdash Y \parallel \Gamma \vdash t : \tau\), writing for \(Y\) the least nominal context that can support \(c\).

Next, we define the syntax of Nominal LCF relative to a signature \(\Sigma\) via a typing judgment \(\Sigma \vdash Y \parallel \Gamma \vdash t : \tau\), presupposing \(Y \text{ nomctx}, \Gamma \text{ varctx}\) and \(\tau \text{ sort}\).

The general rules

\[
\begin{align*}
\Sigma \cdot Y \parallel \Gamma \vdash c : \tau & \quad \text{const} \\
\Sigma \cdot Y \parallel \Gamma \vdash x : \tau & \quad \text{var} \\
\Sigma \cdot Y \parallel \Gamma \vdash \lambda x. t : \tau_1 \Rightarrow \tau_2 & \quad \text{abs} \\
\Sigma \cdot Y \parallel \Gamma \vdash t_1(t_2) : \tau_2 & \quad \text{app} \\
\Sigma \cdot Y \parallel \Gamma \vdash \text{rec}_x \text{ in } t : \tau & \quad \text{rec}
\end{align*}
\]

The const rule embeds the \(\Sigma\)-constants into the syntax; the rec rule shows
how to form self-referential objects.

**Rules for forming tactics and multitactics**

\[
\begin{align*}
\Sigma \vdash Y \parallel \Gamma \vdash t_1 : \text{tac} \\
\Sigma \vdash Y \parallel \Gamma \vdash t_2 : \text{tac} \\
\Sigma \vdash Y \parallel \Gamma \vdash [t_1 \mid t_2] : \text{tac} \\
\Sigma \vdash Y \parallel \Gamma \vdash t_1 : \text{mtac} \quad \text{orelse} \\
\Sigma \vdash Y \parallel \Gamma \vdash \exists \Psi : \text{tele} \quad \text{seq} \\
\Sigma \vdash Y \parallel \Gamma \vdash [t_0, \ldots, t_u] : \text{mtac} \\
\Sigma \vdash Y \parallel \Gamma \vdash t : \text{mtac} \quad i \in \mathbb{N} \\
\Sigma \vdash Y \parallel \Gamma \vdash \diamond t : \text{mtac}
\end{align*}
\]

The seq rule realizes a streamlined modernization of the classic THEN, THENL, THENF family of LCF sequencing tacticals; the distinction between the two forms of sequencing is decomposed using the notion of multitactic, a form of tactic that operates on a full proof state rather than a single goal:

\[
\begin{align*}
\Sigma \vdash Y \parallel \Gamma \vdash t : \text{tac} \\
\Sigma \vdash Y \parallel \Gamma \vdash \Box t : \text{mtac} \quad \text{all} \\
\Sigma \vdash Y \parallel \Gamma \vdash t_i : \text{tac} \quad (i \leq n) \\
\Sigma \vdash Y \parallel \Gamma \vdash [t_{0\parallel}, \ldots, t_n] : \text{mtac} \quad \text{each} \\
\Sigma \vdash Y \parallel \Gamma \vdash \Diamond t : \text{mtac} \quad \text{some}
\end{align*}
\]

Multitactics are sequenced in a *nominal telescope*, a form of vector that binds some number of atoms following each element:

\[
\begin{align*}
\Sigma \vdash Y \parallel \Gamma \vdash \cdot : \text{tele} \quad \text{nil} \\
\Sigma \vdash Y \parallel \Gamma \vdash \cdot \leftarrow m : \text{tele} \quad \text{cons}
\end{align*}
\]

What does it mean to bind hypotheses in the sequencing tactical as in cons above? In the execution of \(\exists \langle a, b, c, \ldots \leftarrow m; \Psi \rangle\), we intend that \(m\) will consume or use as many of the names provided in the binding \(a, b, c, \ldots\) as it needs; in the case of the product elimination rule \(\text{el}_\times[x]\), just \(a\) and \(b\) would be consumed to represent the fresh hypotheses which code the left and right projections of the product, and \(c, \ldots\) would be discarded.

It is important to allow the user to specify “too many” hypothesis names, since it may be that the number required will depend dynamically on the goal at which the tactic script is executed. Furthermore, it should also be possible for the user to provide “too few” hypothesis names, for the same reason; in either case, we must be able to determine exactly how many names were in fact used by a multitactic in a nominal telescope, regardless of how many the user provided. The specification of the exact dynamics of these intentions is the purpose of the denotational semantics given in Section 5.

For the sake of readability, we will often employ the following notational
conventions when writing out nominal telescopes:

\[
\vec{a} \leftarrow m \equiv \vec{a} \leftarrow m; \cdot
\]

\[
m; \Psi \equiv \cdot \leftarrow m; \Psi
\]

**Example 3.1.** Suppose \(\Sigma \vdash a \parallel \text{hyp}[a] : \text{tac}\), \(\Sigma \vdash a \parallel \text{elim}_x[a] : \text{tac}\), and \(\Sigma \vdash \cdot \parallel \text{intro}_x : \text{tac}\), where the first codes a hypothesis rule, the second a product elimination rule, and the third a product introduction rule. Then, we can combine these into a script that eliminates the hypothesis \(a\), generating two fresh hypotheses \(b, c\), and then reconstitutes the product using these two hypotheses:

\[
\exists \langle b, c \leftarrow \square \text{elim}_x[a]; \square \text{intro}_x; [\text{hyp}[b], \text{hyp}[c]] \rangle
\]

The formal derivation of this term is given in Figure 2.

**Symbolic References** So far, hypothesis names have participated in the syntax only as indices to operators and through binding; in a tactic metalanguage, however, it is useful to be able to pass such names as objects. Following Harper [6], we introduce a notion of symbolic reference:

\[
\begin{align*}
\Sigma \vdash Y \parallel \Gamma \vdash a : \text{sym} & \quad \text{quote} \\
\Sigma \vdash Y \parallel \Gamma \vdash h_1 : \text{sym} & \quad \Sigma \vdash Y \parallel \Gamma \vdash t_1 : \tau \\
\Sigma \vdash Y \parallel \Gamma \vdash h_2 : \text{sym} & \quad \Sigma \vdash Y \parallel \Gamma \vdash t_2 : \tau \\
\Sigma \vdash Y \parallel \Gamma \vdash \text{if } h_1 = h_2 \text{ then } t_1 \text{ else } t_2 : \tau & \quad \text{test}
\end{align*}
\]

### 3.1 Encoding rules using symbolic references

In Example 3.1, we saw the encoding of primitive rules that target hypotheses via indexed constants like \(\text{elim}_x[a]\) and \(\text{hyp}[a]\), which embed directly into the tactic language at any context \(\Gamma \varctx\):

\[
\begin{align*}
\Sigma \vdash a \parallel \text{elim}_x[a] : \text{tac} & \quad \text{const} \\
\Sigma \vdash a \parallel \text{hyp}[a] : \text{tac} & \quad \text{const}
\end{align*}
\]

For many logics, this will suffice, but in order to enable more flexible modes of compositionality in which the targets of rules can be computed (Example 3.2), it is also useful to consider an alternative encoding of such rules using symbolic references and the built-in functional sort \(\tau_1 \Rightarrow \tau_2\), as follows:

\[
\begin{align*}
\Sigma \vdash \cdot \parallel \text{elim}_x : \text{sym} & \Rightarrow \text{tac} \\
\Sigma \vdash \cdot \parallel \text{hyp} : \text{sym} & \Rightarrow \text{tac}
\end{align*}
\]

(3.1) (3.2)

Then, we can form a tactic to eliminate the hypothesis \(a\) using the app and
Figure 2: The derivation of Example 3.1.
quote rules, as follows:

$$\begin{align*}
\Gamma \vdash \alpha : \text{sym} & \Rightarrow \text{tac} & \text{(3.1)} \\
\Sigma \vdash \alpha \parallel \Gamma \vdash \alpha \text{: sym} & \Rightarrow \text{app} & \text{(3.3)} \\
\end{align*}$$

One major benefit of this approach is that the hypothesis passed to the elimination rule can be a variable, and thence it is possible to choose a target dynamically; on the other hand, in the case of the indexed operator $\text{elim}_\alpha [\alpha]$, the choice of $\alpha$ must be entirely static.

Example 3.2 (Dynamic hypotheses). Consider the case of a functional tactic that dynamically receives a hypothesis to target as a symbolic reference:

$$\begin{align*}
\text{fst}_\alpha & \triangleq \lambda h. \exists (a \leftarrow \text{elim}_\alpha(h); \Box \text{hyp}'(\alpha')) \\
\text{snd}_\alpha & \triangleq \lambda h. \exists (a, b \leftarrow \text{elim}_\alpha(h); \Box \text{hyp}'(\beta'))
\end{align*}$$

The formal derivation of $\text{fst}_\alpha$ is given in Figure 5. Incidentally, it is to be remarked that only as many names as will be used need to be provided in the binding of the nominal telescope; hence $\text{fst}_\alpha$ need bind only $\alpha$.

4 Comparison of Nominal LCF with LCF

It is easy to see why it is so crucial to be able to determine the exact number of names consumed by a multitactic, as we will do in Section 5: if we could not do this, we would be forced to impose a restriction on tactic scripts which would be devastating to modularity and compositionality, namely to require that names be passed explicitly and manually to every single rule (leaf node), as opposed to generically through the sequencing tactical. This is, in fact, essentially the design used by Coq’s Ltac language, and we consider the Nominal LCF approach to be a significant improvement on it.

To see concretely how the differing policies affect the development of proof scripts, let us consider a simple example of a script that proves the following sequent:

$$p : (((A \times B) \times C) \times D) \vdash A \times (B \times (C \times D)) \text{ true}$$

Figure 3 shows how one might naively implement such a script in standard LCF. Now, this is well and good, but a user will typically try and factor this script into a couple of tactics; for our example, let us consider the development of an LCF tactic to split a left-nested 4-product, shown in Figure 4.

That is essentially fine, but it would be desirable to define this procedure once-and-for-all for left-nested products of any size using recursion. It is not clear how to do this, however, because the tactic needs to take as arguments the
fun script h = 
  let
    val a = fresh "a"
    val b = fresh "b"
    val c = fresh "c"
    val d = fresh "d"
    val abc = fresh "abc"
    val ab = fresh "ab"
  in
    ProdElim h (abc, d) THEN
    ProdElim abc (ab, c) THEN
    ProdElim ab (a, b) THEN
    ProdIntro THENL
    [Hyp a, ProdIntro THENL
     [Hyp b, ProdIntro THENL
      [Hyp c, Hyp d]]]
  end

Figure 3: A naïve LCF script to reassociate a four-product.

val ProdIntros = 
  FIX (fn t => TRY (ProdIntro THEN t))

fun SplitProd4 h (a, b, c, d) = 
  let
    val abc = fresh "abc"
    val ab = fresh "ab"
  in
    ProdElim h (abc, d) THEN
    ProdElim abc (ab, c) THEN
    ProdElim ab (a, b)
  end

fun script' h = 
  let
    val a = fresh "a"
    val b = fresh "b"
    val c = fresh "c"
    val d = fresh "d"
  in
    ProdElim h (a, b, c, d) THEN
    ProdIntros THENL
    [Hyp a, Hyp b, Hyp c, Hyp d]
  end

Figure 4: A refactoring of the LCF script in Figure 3.
Figure 5: The derivation of \( \text{fst}_x \) in Example 3.2.
names it will use—but the number of names needed will depend on the goal!

One might resolve this by taking a list of names as an argument, and recursing through this list simultaneously with the goal, but this is an error-prone, low-level technique to which the user should not be forced to resort; in fact, it is this very solution which Nominal LCF implements once and for all with its sequencing tactical, but in a more well-behaved way than would be feasible for a user of LCF to implement in a development.

For comparison, what follows is the corresponding Nominal LCF script, factored through a generic tactic for decomposing a left-nested product of arbitrary length:

\[
\begin{align*}
\text{try} & \triangleq \lambda t. (t \mid \text{id}) \\
\text{intros}_x & \triangleq \text{rec } t \text{ in } \text{try}(\exists \langle \text{intro}_x; t \rangle) \\
\text{splits}_x & \triangleq \text{rec } t \text{ in } \lambda h. \text{try}(\exists \langle a, b \leftarrow \text{elim}_x(h); t(a) \rangle) \\
\text{script} & \triangleq \lambda h. \exists \langle a, b, c, d \leftarrow \text{splits}_x(h); \\
& \quad \text{intro}_x; \\
& \quad \text{intros}_x; \\
& \quad \text{hyp}(\text{'a}), \text{hyp}(\text{'b}), \text{hyp}(\text{'c}), \text{hyp}(\text{'d}) \rangle
\end{align*}
\]

5 Denotational Semantics of Nominal LCF

We will explain the meaning of the Nominal LCF language by interpreting it into a theory of dI-domains, which is developed in Appendix A. The purpose of using domains is twofold:

1. To interpret recursion, which is a crucial part of tactic scripts.
2. To make precise the notion of tactic continuity, which will be used in the execution of the sequencing tactical.

Why denotational semantics? Is it really necessary to use a denotational semantics to explain Nominal LCF rather than an operational semantics? Based on point (2) above, we can answer in the affirmative.

To execute the sequencing tactical, we need to know exactly how many bound hypothesis names are in fact consumed by a tactic; in the case of primitive rules, this can usually be determined uniformly and statically, but in the presence of complex tacticals such as \( t_1 | t_2 \) and also recursive tactics \( \text{rec } x \text{ in } t[x] \), this is a purely dynamic property that must be calculated on the basis of a continuity theorem for tactics, which is established by semantical methods.

5.1 LCF Languages

An LCF language for a signature \( \Sigma \) consists in a monad \((\mathcal{G}, \eta, \mu)\) on \( \text{Dom}_\Sigma \), the category of dI-domains (see Appendix A) and stably continuous functions. \( \mathcal{G}(D) \)
shall represent proof states taking goals in $D$, or partial proof trees whose leaves are goals; the natural transformation $\eta : 1 \rightarrow \mathcal{E}$ initiates a proof state with a goal, whereas $\mu : \mathcal{E} \circ \mathcal{E} \rightarrow \mathcal{E}$ implements the substitution of new proof states for the existing goals. We will write $f^* : \mathcal{E}(D) \rightarrow \mathcal{E}(E)$ for the Kleisli extension $\mu \circ \mathcal{E}(f)$ of $f : D \rightarrow \mathcal{E}(E)$; in programming languages like Haskell and ML, this operation is usually called $\text{bind}$.

We also require a semigroup structure on $\mathcal{E}$, i.e. an associative natural transformation $+ : \mathcal{E} \times \mathcal{E} \rightarrow \mathcal{E}$, which will be used for choosing between a “failing” and a “succeeding” proof state.

Additionally, an LCF language shall induce a natural transformation $\pi : \mathcal{E} \rightarrow \text{List}$, which should be thought of as projecting the goals from a proof state; moreover, where $\text{label} : \text{List} \rightarrow \text{List}(\mathbb{N} \times -)$ is the transformation that labels the elements of a list by their index, there shall exist a corresponding transformation $\tilde{\text{label}} : \mathcal{E} \rightarrow \mathcal{E}(\mathbb{N} \times -)$ such that the following diagram commute:

$$
\begin{array}{ccc}
\mathcal{E} & \xrightarrow{\text{label}} & \mathcal{E}(\mathbb{N} \times -) \\
\downarrow{\pi} & & \downarrow{\pi} \\
\text{List} & \xrightarrow{\text{label}} & \text{List}(\mathbb{N} \times -)
\end{array}
$$

Functions $f : D \rightarrow \mathcal{E}(D)$ will be called $D$-tactics, and endofunctions $f : \mathcal{E}(D) \rightarrow \mathcal{E}(D)$ will be called $D$-multitactics.

5.2 The definition of a $\Sigma$-model

Let $D^w$ be the domain of streams or choice sequences of $D$s, ordered by approximation. Let $\mathbb{A}$ be the infinite domain of nominal atoms; then a stream of atoms is a member of the domain $\mathbb{A}^w$; we will use this to model the source of names for hypotheses used in the Nominal LCF dynamics.

A $\Sigma$-model $M$ consists in an LCF language $\mathcal{L}$, as well as a distinguished domain $\mathcal{J}$ of “judgments” such that $\mathcal{E}(\mathcal{J})$ is flat (Definition A.4, p. 17), along with an interpretation $M[\llbracket c : \tau \rrbracket]$ of every primitive constant $\Sigma \vdash \tau \parallel c : \tau$ into the semantic domain $M[\llbracket \tau \text{ sort} \rrbracket]$, defined as follows:

$$
\begin{align*}
M[\llbracket \tau \text{ sort} \rrbracket] & \triangleq \mathbb{A}^w \rightarrow M[\llbracket \tau \text{ sort} \rrbracket] \\
M[\llbracket \text{tac sort} \rrbracket] & \triangleq \mathcal{J} \rightarrow \mathcal{E}(\mathcal{J}) \\
M[\llbracket \text{ntac sort} \rrbracket] & \triangleq \mathcal{E}(\mathcal{J}) \rightarrow \mathcal{E}(\mathcal{J}) \\
M[\llbracket \text{tele sort} \rrbracket] & \triangleq \mathcal{E}(\mathcal{J}) \rightarrow \mathcal{E}(\mathcal{J}) \\
M[\llbracket \text{sym sort} \rrbracket] & \triangleq \mathbb{A} \\
M[\llbracket \tau_1 \Rightarrow \tau_2 \text{ sort} \rrbracket] & \triangleq M[\llbracket \tau_1 \text{ sort} \rrbracket] \rightarrow M[\llbracket \tau_2 \text{ sort} \rrbracket]
\end{align*}
$$

The interpretation of sorts above factors through an intermediate interpretation $M[\llbracket \tau \text{ sort} \rrbracket]$, which can be thought of as the \textit{values} of sort $\tau$, with $M[\llbracket \tau \text{ sort} \rrbracket]$
being the stream computations of sort $\tau$. Functional abstraction is crucially lazy, in the sense that the input to a function is a stream computation, not a value.

5.3 The Modulus of Continuity Functional

In order to define the interpretation of Nominal LCF into a $\Sigma$-model in Section 5.4, it will be necessary to first develop a continuity theorem for multitactics, namely that for any nominal multitactic $F \in \mathcal{M}$ and $\alpha \in \mathcal{A}$, $S \in \mathcal{G}(J)$, there is a least number of elements of $\alpha$ consumed in order to compute $F(\alpha)(S)$.

By Theorem B.1 (p. 19) and Theorem B.2 (p. 19), we assert the existence for each $F$ of a continuous function $M(F) \in \mathcal{A} \times \mathcal{S}(J) \to \mathbb{N}_\bot$ called the modulus of continuity, which computes exactly this least prefix.

5.4 Interpretation of Nominal LCF into a $\Sigma$-model

A $\Sigma$-model can be extended to interpret the full syntax of Nominal LCF in a straightforward way. To each form of syntax $\Sigma \triangleright \Upsilon \parallel \Gamma \vdash t : \tau$, we give an interpretation $\mathcal{M}[\tau \triangleright \Upsilon \parallel \Gamma \vdash t : \tau]_{\rho}$ in the domain $\mathcal{M}[\tau \triangleright \Upsilon \parallel \Gamma \vdash t : \tau]_{\rho}$, relative to a valuation (environment) $\rho(x) \in \mathcal{M}[\tau \triangleright \Upsilon \parallel \Gamma \vdash t : \tau]_{\rho}(x : \tau \in \Gamma)$ of the free variables in context.

**Interpretation of general rules**

$$M[\mathcal{x} : \tau]_{\rho} \triangleq \rho(x) \quad \text{(var)}$$

$$M[\mathcal{c} : \tau]_{\rho} \triangleq M[c : \tau] \quad \text{(const)}$$

$$M[\lambda x. t(x) : \tau_1 \Rightarrow \tau_2]_{\rho} \triangleq \lambda\alpha.\lambda T. M[t(x) : \tau_1]_{\rho, x \mapsto \alpha}(a) \quad \text{(abs)}$$

$$M[t_1(t_2) : \tau_2]_{\rho} \triangleq \lambda a. \left( M[t_1 : \tau_1 \Rightarrow \tau_2]_{\rho}(a) \right) \left( M[t_2 : \tau_1]_{\rho} \right) \quad \text{(app)}$$

$$M[\mathcal{rec} \ x \ in \ t[x] : \tau]_{\rho} \triangleq \text{fix}(\lambda T. M[t : \tau]_{\rho, x \mapsto T}) \quad \text{(rec)}$$

**Interpretation of symbolic references**

$$M[\mathcal{\text{\textasciitilde}a} : \text{sym}]_{\rho} \triangleq \lambda \alpha. a \quad \text{(quote)}$$

$$M[\text{if } h_1 = h_2 \ then \ t_1 \ else \ t_2 : \tau]_{\rho} \triangleq \lambda \alpha. \begin{cases} a_1 = M[h_1 : \text{sym}]_{\rho}(\alpha) \\ a_2 = M[h_2 : \text{sym}]_{\rho}(\alpha) \end{cases} \quad \text{let}$$

$$M[\text{if } h_1 = h_2 \ then \ t_1 \ else \ t_2 : \tau]_{\rho} \triangleq \lambda \alpha. \begin{cases} M[t_1 : \tau]_{\rho}(\alpha), \ &\text{if } a_1 = a_2 \\ M[t_2 : \tau]_{\rho}(\alpha), \ &\text{if } a_1 \neq a_2 \end{cases} \quad \text{in}$$

**Interpretation of tactics** We begin with the interpretation of syntactic tactics as nominal tactics. In order to interpret seq, we appeal to an auxiliary interpretation
of nominal telescopes as iterated compositions of multitactics.

\[
M[\langle t_1 \mid t_2 : \text{tac} \rangle] \triangleq \lambda \alpha. \lambda J. M[\langle t_1 : \text{tac} \rangle]_\rho(\alpha)(J) + M[\langle t_2 : \text{tac} \rangle]_\rho(\alpha)(J) \quad \text{(orelse)}
\]

\[
M[\exists \Psi : \text{tac}]_\rho \triangleq \lambda \alpha. M[\langle \Psi : \text{tele} \rangle]_\rho(\alpha) \circ \eta \quad \text{(seq)}
\]

**Interpretation of telescopes** Let \(\vec{a}@\alpha\) be the stream obtained from \(\alpha\) by prepending the finite sequence \(\vec{a}\); let \(\alpha \setminus n\) be the stream obtained by removing the first \(n\) elements of \(\alpha\). Now, we may proceed to interpret the iterated composition of a nominal telescope of multitactics:

\[
M[\cdot : \text{tele}]_\rho \triangleq \lambda \alpha. \lambda S. S
\]

\[
M[\vec{a} \leftarrow m; \Psi : \text{tele}]_\rho \triangleq \begin{cases} 
M(\alpha) = M[\langle m : \text{mtac} \rangle]_\rho(\vec{a}@\alpha) \\
\mu(\alpha, S) = \max(0, M(M)(\vec{a}@\alpha, S) - \text{len}(\vec{a})) \\
M'(\alpha, S) = M[\langle \Psi : \text{tele} \rangle]_\rho(\alpha \setminus \mu(\alpha, S)) 
\end{cases}
\]

\[\text{in } \lambda \alpha. \lambda S. M'(\alpha, S)(M(\alpha, S)) \quad \text{(cons)}\]

The entire Nominal LCF apparatus hinges upon the interpretation of the cons rule above. When a vector of names \(\vec{a}\) is bound in a telescope of multitactics \(\vec{a} \leftarrow m; \Psi\), it is pushed onto the front of the name stream \(\alpha\) that is consumed by the denotation of \(m\).

Note that \(m\) may in fact consume names from \(\vec{a}@\alpha\) beyond those merely in \(\vec{a}\); in this case, making crucial use of the modulus of continuity developed in Section 5.3, we take the total number of names used by \(m\) and subtract from it the length of \(\vec{a}\), and pop this many names off the front of \(\alpha\) before feeding it to the denotation of \(\Psi\).

**Remark 5.1.** It is important to remember that whilst we have not shown \(M(\cdot)\) to continuously transform a functional to its modulus of continuity, the resulting modulus of continuity \(M(M)\) itself is a continuous function (Theorem B.2). As a result, no discontinuity arises from using \(M(M)\) in the definition of \(M'\), since the calculation is metatheoretical in nature.
Interpretation of multitactics  Finally, the multitactics are interpreted using the Kleisli extension \((-)^*\) and the \texttt{label} transformation that labels subgoals with their indices:

\[
M \llbracket \Box t : \text{mtac} \rrbracket \triangleq \lambda \alpha. \left( M \llbracket t : \text{tac} \rrbracket (\alpha) \right)^* \quad \text{(all)}
\]

\[
M \llbracket [i] t : \text{mtac} \rrbracket \triangleq \lambda \alpha. \left( \lambda (i, J). M \llbracket [i] : \text{tac} \rrbracket (\alpha)(J) \right)^* \circ \texttt{label} \quad \text{(each)}
\]

\[
M \llbracket \Diamond t : \text{mtac} \rrbracket \triangleq \lambda \alpha. \left( \lambda (i, J). \begin{cases} M \llbracket t : \text{tac} \rrbracket (\alpha)(J), & \text{if } i = j \\ \eta(J), & \text{if } i \neq j \end{cases} \right)^* \circ \texttt{label} \quad \text{(some)}
\]

This concludes the interpretation of the Nominal LCF language into a $\Sigma$-model.

6 Related work

Nominal Second-Order Algebra  The notion of a signature to a second-order algebraic theory which is fibred over nominal contexts was first introduced by Robert Harper in his textbook, Practical Foundations for Programming Languages [6], and endowed with a denotational semantics by Sterling and Morrison in their unpublished note, Syntax and Semantics of Abstract Binding Trees [16].

The syntax presented in the present paper is readily formalized using Harper’s abstract binding tree (abt) logical framework; the Standard ML implementation of Nominal LCF is itself also based on abts.

Brouwer’s free choice sequences  The algebraic cpos (Definition A.5) of Scott’s Domain Theory [14] are essentially a generalization of Brouwer’s notion of a spread [1, 18, 3, 17], whose finite approximations form the compact basis for a collection of maximal elements (choice sequences). The stream domain $D^\omega$, as defined in this paper, is precisely an example of Brouwer’s dressed spreads, and we are justified in calling the maximal elements of $A^\omega$ choice sequences of atoms.

Choice of hypothesis names is not the only interactive phenomenon that has been explained using choice sequences; for instance, Muller et al. have devised a Kripke logical relation in which observational independence of interactive phenomena is modeled using invariance over the points (choice sequences) of the universal spread [9]. Moreover, Escardó describes an extension to Gödel’s System T with an oracle, which is interpreted relative to a choice sequence $\alpha$, and uses this to prove the uniform continuity of all T-definable functionals on the Cantor space [4].
Acknowledgments

The author would like to thank Mark Bickford and Andrej Bauer for illuminating conversations on continuity; Robert Harper for discussions concerning symbols and symbolic references in the syntax of programming languages; and Darin Morrison for his many suggestions which helped to improve this paper. All mistakes belong to the author.
A Domain Theory

The present section develops the machinery necessary to define a well-behaved (Cartesian closed) category of domains, which is used in our interpretation.

Definition A.1 (complete partial orders). A dcpo (directed-complete partial order) is a poset $D \equiv (\mathcal{D}, \sqsubseteq)$ such that every directed subset $U \subseteq \mathcal{D}$ has a least upper bound (lub, supremum) $\bigsqcup U$. We say that $U$ is directed when it is non-empty, and every finite subset $V \subseteq U$ has an upper bound [5]. Moreover, we say that a dcpo is pointed when it has a least element, usually written $\bot_u\zeta$; a pointed dcpo is called a cpo.

Definition A.2 (Scott continuity). We say that a map $F \in \mathcal{D} \to \mathcal{E}$ is Scott-continuous when $F$ preserves suprema on directed subsets. Formally:

$$\forall M \subseteq \mathcal{D}. M \text{ directed} \implies \bigsqcup F[M] = F(\bigsqcup M)$$

where

$$F[M] \triangleq \{ F(x) \mid x \in M \}$$

Definition A.3 (compact elements). An element of cpo $(\mathcal{D}, \sqsubseteq, \bot_u\zeta)$ is compact when every directed subset whose lub it approximates contains an element which it also approximates [5]. We use the notation $\mathcal{D}^\flat$ for the set of compact elements of $\mathcal{D}$. Formally, for $x \in \mathcal{D}$, we have:

$$\forall M \subseteq \mathcal{D}. M \text{ directed} \land x \sqsubseteq \bigsqcup M \implies \exists m \in M. x \sqsubseteq m$$

where $x \in \mathcal{D}^\flat$

Intuitively, compactness says that an object’s data is finitary—which is not to say that the object in-itself is finitary, but only that it is finitarily generated. For $x \in \mathcal{D}$, let the principal ideal $[x]$ of $x$ be the set of compact elements which approximate it:

$$[x] \triangleq \{ y \in \mathcal{D}^\flat \mid y \sqsubseteq x \}$$

Definition A.4 (flat cpos). A cpo is flat when the order on its non-bottom elements is discrete, i.e. any non-bottom $x$ and $y$ are not comparable. An example of a flat cpo is the natural numbers $\mathbb{N}_\perp$, where only $\perp \subseteq n$ for each $n \in \mathbb{N}$.

Theorem A.1. All elements of a flat cpo $\mathcal{D}_\perp$ are compact.

Proof. Fix an element $x \in \mathcal{D}_\perp$ and a directed subset $M \subseteq \mathcal{D}_\perp$ such that $x \sqsubseteq \bigsqcup M$. We have to show that there exists some $m \in M$ such that $x \sqsubseteq m$; because $\mathcal{D}_\perp$ is flat, there are three possibilities:

1. Case $M \equiv \{ y \}$ for some $y \neq \perp$: choose $m \triangleq y$.
2. Case $M \equiv \{ y, \perp \}$ for some $y \neq \perp$: choose $m \triangleq y$. 

17
3. Case $M \equiv \{ \bot \}$: choose $m \equiv \bot$.

In all three cases, because $\bigsqcup M = m$, we have $x \sqsubseteq m$.

Now we are in a position to define a notion of cpo which is generated by a
finite basis of compact elements.

**Definition A.5** (algebraic cpos). When all elements of a cpo $D$ can be understood
as being generated in a precise way by the compact elements in $D^\circ$, we say that $D$
is algebraic [5]. In particular, each element of $D$ must be the least upper bound of its
principal ideal, which must be directed:

$$\forall x \in D. \{x\} \text{ directed } \wedge x = \bigsqcup \{x\}$$

$D$ algebraic

One useful fact concerning algebraic cpos is that continuous functions be-
tween them are completely determined by their action on compact elements.
Algebraic cpos do not on their own form a Cartesian closed category, because
exponentiation does not preserve algebraicity. We will need to impose a further
constraint:

**Definition A.6** (bounded-complete cpos). An cpo $D$ is bounded-complete when it
contains a suprema for all its bounded subsets [5], in the following sense:

$$\exists x \in U. \forall y \in U. y \sqsubseteq x$$

$U \subseteq D$ bounded

$$\forall U \subseteq D \text{ bounded}, \exists x \in D. x = \bigsqcup U$$

$D$ bounded-complete

**Definition A.7** (distributive domains). A distributive domain is a bounded-complete
algebraic cpo in case suprema distribute over infima. In other words:

$$\forall x, y, z \in D. x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$$

$D$ distributive

**Definition A.8** (dl-domains and stable functions). A dl-domain is a distributed,
bounded-complete algebraic cpo such that the principal ideal of each compact element is
a finite set. A stable function between dl-domains is a continuous function $f \in D \to E$
for which application distributes over infima [5]:

$$f \text{ continuous } \forall x, y \in D. f(x \sqcap y) = f(x) \sqcap f(y)$$

$f$ stable

We order stable functions as follows:

$$\forall x, y \in D. x \sqsubseteq y \Rightarrow f(x) = f(y) \sqcap g(x)$$

$f \sqsubseteq g$
Together, dI-domains and stable functions form the Cartesian closed category $\text{Dom}_s$; for the remainder of this paper, we will refer to objects of $\text{Dom}_s$ simply as domains.

## B Stream continuity for multitactics

**Theorem B.1** (Stability implies stream continuity). Any nominal multitactic $F \in \mathcal{M}[\text{mtac sort}]$ is continuous in the following sense:

$$\forall \alpha \in A^\omega. \forall S \in \mathcal{G}(J). \exists k \in \mathbb{N}. \forall \beta \in A^\omega. \alpha(k) = \beta(k) \implies F(\alpha)(S) = F(\beta)(S)$$

**Proof.** Let $S' \triangleq F(\alpha)(S)$. Stable functions between dI-domains obey a minimal data constraint, such that for any $(\alpha, S) \in F^{-1}(S')$, we can calculate the least compact approximation of $(\alpha, S)$ needed to compute $S'$ assuming that $S' \in \mathcal{G}(J)^\flat$, which we will write $M_F(\alpha, S) \triangleq (\vec{u}, S_0)$. $S'$ is indeed compact because all elements of a flat domain are compact (Theorem A.1). Thus, we can exhibit $k$ as the length of the list $\vec{u}$.

The proof of Theorem B.1 supplies us with a way to calculate this $k$ in our metatheory for each $(F, \alpha, S)$, but it does not obviously give us a continuous function $A^\omega \times \mathcal{G}(J) \to \mathbb{N}$ for each $F$. Our suspicion is based on the observation that taking the length of an arbitrary stream approximation is not generally continuous, because it violates monotonicity. However, we can in fact show that following operation is Scott-continuous for each $F$:

$$M(F)(-) \triangleq \text{len}(\pi_1(M_F(-)))$$

**Theorem B.2** (Continuity of the modulus). For each nominal multitactic $F$, the operation $M(F)(-)$ is Scott-continuous.

**Proof.** Fixing a directed subset $Q \subseteq A^\omega \times \mathcal{G}(J)$, it suffices to show:

$$\bigsqcup M(F)(Q) = M(F) \left( \bigsqcup Q \right)$$

Let $l \triangleq M(F)(\bigsqcup Q)$ and fix $p \in Q$; because the codomain $\mathbb{N}$ is flat, we only need to show that if $M(F)(p) \equiv k$ for $k \neq \bot$, then $k = l$. Because $M_F$ is monotone and $p \subseteq \bigsqcup Q$, we know $M_F(p) \subseteq M_F(\bigsqcup Q)$; therefore, $k$ must be less than or equal to $l$, and it remains only to show that $k$ cannot be less than $l$. But $l$ is the shortest prefix of $\pi_1(\bigsqcup Q)$ that suffices, and therefore $k$ cannot be less than $l$. As a result, we have $k = l$, whence $M(F) : A^\omega \times \mathcal{G}(J) \to \mathbb{N}$ is Scott-continuous at each $F$. 

19
References

[1] L. Brouwer. *Brouwer’s Cambridge Lectures on Intuitionism*. Cambridge University Press, 1981. Edited by Dirk van Dalen. 15

[2] R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986. 1, 3

[3] M. Dummett. *Elements of intuitionism*, volume 39 of *Oxford Logic Guides*. The Clarendon Press Oxford University Press, New York, second edition, 2000. 15

[4] M. H. Escardó. Continuity of Gōdel’s System T definable functionals via effectful forcing. *Electronic Notes in Theoretical Computer Science*, 298:119–141, 2013. 15

[5] C. A. Gunter. *Semantics of programming languages – structures and techniques*. Foundations of computing. MIT Press, 1993. 17, 18

[6] R. Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, New York, NY, USA, 2016. 6, 15

[7] J. Longley. When is a functional program not a functional program? In *Proceedings of Fourth ACM SIGPLAN International Conference on Functional Programming*, pages 1–7. ACM Press, 1999. 3

[8] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0. 1

[9] S. Muller, U. Acar, W. Duff, and R. Harper. Interactive computation in an open world. *http://repository.cmu.edu/compsci/2643/*, 2015. 15

[10] T. Nipkow, M. Wenzel, and L. C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002. 1

[11] L. C. Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge University Press, New York, NY, USA, 1987. 1

[12] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013. 4

[13] R. Pollack. On extensibility of proof checkers. In *Dybjer, Nordström and Smith (eds), Types for Proofs and Programs: International Workshop TYPES’94, Bastad*, pages 140–161. SpringerVerlag, 1995. 1
[14] D. Scott. ISILC Logic Conference: Proceedings of the International Summer Institute and Logic Colloquium, Kiel 1974, chapter Data types as lattices, pages 579–651. Springer Berlin Heidelberg, Berlin, Heidelberg, 1975. 15

[15] J. Sterling, D. Gratzer, and V. Rahli. RedPRL – the People’s Refinement Logic. http://www.jonprl.org/, 2016. 1, 3

[16] J. Sterling and D. Morrison. Syntax and Semantics of Abstract Binding Trees. http://arxiv.org/abs/1601.06298, 2015. 4, 15

[17] A. S. Troelstra and van Dalen, Dirk. Constructivism in mathematics: an introduction. Volume I. Studies in logic and the foundations of mathematics. North-Holland, Amsterdam, New-York, Oxford, 1988. 15

[18] M. van Atten. On Brouwer. Wadsworth Philosophers Series. Thompson/Wadsworth, Toronto, Canada, 2004. 15