Open Proofs and Open Terms: a Basis for Interactive Logic

Herman Geuvers[1] and G. I. Jojgov[2]

[1] University of Nijmegen, The Netherlands
herman@cs.kun.nl
[2] Eindhoven University of Technology, The Netherlands
G.I.Jojgov@tue.nl

Abstract. In the process of interactive theorem proving one often works with incomplete higher order proofs. In this paper we address the problem of giving a correctness criterion for these incomplete proofs by presenting open higher order logic o-Predω as a conservative extension of Predω. We define a typing system o-λPredω and show that the Curry-Howard embedding extends to embedding into o-λPredω. We show that the contexts of this typing system extended with definitions can represent the notion of proof state and allow for forward reasoning, proof reuse and free exploration of the given data ('stratch paper' mechanism).

1 Introduction

Logic is about *finished* proofs and not about the process of *finding* a proof. The derivation rules of a logic define inductively what is *derivable*. The rules do not tell us how we should *find* or *construct* such a derivation, but they give us a procedure of *checking* whether an alleged proof is indeed well-formed. Of course, the derivation rules are chosen in such a way that they represent 'obviously correct' reasoning steps, but that does not mean that mathematicians actually reason in this way. The typical situation is that when one tries to prove a result, one makes intermediates claims, leaves parts temporarily unspecified and freely explores the possibilities. Then, when the proof is 'finished', it is written up in a style that corresponds - at least in spirit - to natural deduction. If we look more closely at the process of proof finding, we observe that also in that phase, the proof-steps are intended to be correct in terms of natural deduction. So, there should be a correctness criterion for 'unfinished proofs', where some parts may be left open or unspecified, but the steps that have been made are correct. One place where these unfinished proofs appear is in systems for *theorem proving*, where the computer helps to verify theorems.

Theorem proving systems for mathematics can be *interactive* or *batch-oriented*. In the first case the user types in *tactics* that guide the system through the proof-construction (or, if the system does not really construct a proof, through the verification of the theorem in the logic of the system). In the second case the user inputs a complete proof and the system works as a *proof-checker* that checks whether all the proof-steps are correct. Systems like Automath and Mizar are batch-oriented. It may seem that the user has to provide quite a lot of details for such a system to be able to verify a proof. (Of course this depends on the type of proof-steps that the system can infer itself.) Mizar [1] shows that one can formalize large parts of mathematics in such a system. In interactive systems, the amount of detail that has to be provided by the user depends on the power of the tactics. There is another issue that is particularly important for interactive systems: how to communicate to the user what the present *state of the proof* is, in order for the user to make a sensible next step. This is especially important if we want to use a theorem prover for trying to prove a new result or for freely exploring the mathematics. (At this moment, theorem provers are not suited for assisting in proving new results: one formalizes existing proofs.) But also if one tries to formalize an existing proof, the system needs to communicate the *proof state* to the user, in the form of an unfinished proof. More in general, it is important to understand precisely what these interactive theorem provers actually operate on. So we want to give a precise meaning to 'unfinished proofs' and 'proof states'.

Important questions that arise when looking at the process of proof finding and at interactive theorem proving are:

– Can we give a correctness criterion for unfinished proofs? (In such a way that many of the existing 'open proofs' are captured.)
– Can we give a correctness criterion for operations on unfinished proofs? (In such a way that known tactics are instances of such operations.)

So, we first have to answer the questions what an *unfinished proof* and what a *proof state* are. The way mathematicians (and others) give their proofs closely represents – at least in spirit – natural deduction. Hence, if we want to formalize the notion of unfinished proof, natural deduction is a good starting point. So, then the question is: what is an *unfinished natural deduction*? And what are correct *operations* on these unfinished natural deductions? In this paper we will mainly be answering the first question, taking inspiration from the second one, because we know – intuitively and from experience with interactive theorem provers – quite well what we want to be able to do.

In the following section we take natural deduction for higher order predicate logic as a starting point and treat a number of examples of 'open proofs' and how we might want to operate on them. These will serve as a motivation for the rest of the paper. The examples are chosen to be quite trivial, which is done deliberately to keep the exposition small and to be able to pinpoint at the crucial issues. Then we define *open higher order predicate logic*, a version of higher order predicate logic where we allow unfinished (open) proofs and open terms. We discuss to which extent this captures the examples. Finally we define a type theoretic variant of this open higher order predicate logic. We extend the well-known formulas-as-types embedding to include open proofs and open terms, yielding a formulas-as-types embedding from open higher order logic to this type theory. Then we show how this type theory nicely captures the notion of *proof state*, which is now a *context* in this type theory.

2 Motivating Examples

1. An unfinished proof with *backward* proof construction.
We start with the goal of proving $A\rightarrow C$ from hypotheses $A\rightarrow B\rightarrow C$ and $A\rightarrow B$ (1). We solve this goal by the rule for introduction of implication (2). This introduces a new hypothesis A. In (3) we have used the hypothesis $A\rightarrow B\rightarrow C$ to deduce C by implication-elimination of the new goals A and B. The first one is solved in (4) by the assumption A and the second by introducing a new goal A and eliminating the assumption $A\rightarrow B$. Finally (5) we solve A trivially by the hypothesis A and we have a complete derivation of $A\rightarrow C$ from $A\rightarrow B\rightarrow C$ and $A\rightarrow B$.

1. $A\rightarrow B\rightarrow C \quad A\rightarrow B$

$$\dfrac{?}{A\rightarrow C}$$

2. $A\rightarrow B\rightarrow C \quad A\rightarrow B \quad [A]^i$

$$\dfrac{\dfrac{?}{C}}{A\rightarrow C}\;{}^{i}$$

3. $A\rightarrow B\rightarrow C \quad A\rightarrow B \quad [A]^i$

$$\dfrac{\dfrac{\dfrac{\dfrac{\checkmark}{A\rightarrow B\rightarrow C} \quad \dfrac{?}{A}}{B\rightarrow C} \quad \dfrac{?}{B}}{C}}{A\rightarrow C}\;{}^{i}$$

4. $A\rightarrow B\rightarrow C \quad A\rightarrow B \quad [A]^i$

$$\dfrac{\dfrac{\dfrac{\dfrac{\checkmark}{A\rightarrow B\rightarrow C} \quad \dfrac{\checkmark}{A}}{B\rightarrow C} \quad \dfrac{\dfrac{\checkmark}{A\rightarrow B} \quad \dfrac{?}{A}}{B}}{C}}{A\rightarrow C}\;{}^{i}$$

5. $A\rightarrow B\rightarrow C \quad A\rightarrow B \quad [A]^i$

$$\dfrac{\dfrac{\dfrac{\dfrac{\checkmark}{A\rightarrow B\rightarrow C} \quad \dfrac{\checkmark}{A}}{B\rightarrow C} \quad \dfrac{\dfrac{\checkmark}{A\rightarrow B} \quad \dfrac{\checkmark}{A}}{B}}{C}}{A\rightarrow C}\;{}^{i}$$

2. An unfinished proof with a *forward* proof construction.

We proceed forward by using elimination rules on the hypotheses. In (3) we have used the A and $A\rightarrow B$ to obtain B which is used in (4) to deduce $B\rightarrow C$. Then we must infer B again and use it to derive C at step (4).

Note that in step (4) we would like to be able to *reuse* the already proven result B instead of having to derive it again, but natural deduction does not allow this.

1. $B\rightarrow B\rightarrow C \quad A\rightarrow B \quad A$

$$\dfrac{?}{C}$$

2. $B\rightarrow B\rightarrow C \quad \dfrac{A\rightarrow B \quad A}{B}$

$$\dfrac{?}{C}$$

3. $\dfrac{B\rightarrow B\rightarrow C \quad \dfrac{A\rightarrow B \quad A}{B}}{B\rightarrow C}$

$$\dfrac{?}{C}$$

4. $\dfrac{\dfrac{B\rightarrow B\rightarrow C \quad \dfrac{A\rightarrow B \quad A}{B}}{B\rightarrow C} \quad \dfrac{A\rightarrow B \quad A}{B}}{C}$

3. An unfinished proof with *open terms*
In this example we have a transitive relation $R(x,y)$ and we want to prove $R(a,c)$. The question is *what to take for y?* We don't know (yet), so we want to leave y *open.*

From this example we see that open terms arise quite naturally in interactive theorem proving if we want to postpone the specific choice of a value for a variable.

1. $\forall x,y,z.R(x,y)\rightarrow R(y,z)\rightarrow R(x,z)$

$$\dfrac{?}{R(a,c)}$$

2. $\forall x,y,z.R(x,y)\rightarrow R(y,z)\rightarrow R(x,z)$

$$\dfrac{\dfrac{\overset{\checkmark}{R(a,\underline{y})\rightarrow R(\underline{y},c)\rightarrow R(a,c)} \quad \overset{?}{R(a,\underline{y})}}{R(\underline{y},c)\rightarrow R(a,c)} \quad \dfrac{?}{R(\underline{y},c)}}{R(a,c)}$$

The 'open place' $\underline{y}$ in the example has a different role than a variable: we seek an value for it and we will not abstract over it. We will call these variables *meta-variables.* A term containing a meta-variable will be called an *open term.*

Convention 1 *To clearly distinguish variables from meta-variables, we will underline meta-variables, so $\underline{y}$ denotes a meta-variable and y is different from $\underline{y}$.*

4. Delaying the choice for the *witness* for an existential quantifier and *computing* with open terms.
In this example, the meta-variable $\underline{n}$ should actually *depend* on y, because we want to be able to instantiate $\underline{n}$ with y. If we do that in the last proof (4), y becomes an unbound variable, so that is not correct. Hence we have to be careful with the definition of instantiation. As we can see, the problem occurs because *reduction* and *instantiation* do not *commute.*

1. $\dfrac{?}{\exists f\forall x.f(x)=x}$

2. $\dfrac{\dfrac{?}{\forall x.\underline{f}(x)=x}}{\exists f\forall x.f(x)=x}$

3. $\dfrac{\dfrac{\dfrac{?}{\forall x.(\lambda y.\underline{n})(x)=x}}{\forall x.(\lambda y.\underline{n})(x)=x}}{\exists f\forall x.f(x)=x}$

4. $\dfrac{\dfrac{\dfrac{\dfrac{???}{\forall x.\underline{n}=x}}{\forall x.(\lambda y.\underline{n})(x)=x}}{\forall x.(\lambda y.\underline{n})(x)=x}}{\exists f\forall x.f(x)=x}$

To prove the correctness of instantiation, we would need a more general property (Lemma 14), namely that instantiation must commute with the derivation rules. This property is depicted in the following diagram, which is given together with its instance to the above example (where it fails).

$$\begin{tikzcd}
M \arrow[r, "\text{instantiate } \underline{n}:=t"] \arrow[d, "\beta"] & N \arrow[d, "\beta"'] \\
P \arrow[r, "\text{instantiate } \underline{n}:=t"] & ??
\end{tikzcd}
\qquad
\begin{tikzcd}
(\lambda y.\underline{n})(x) \arrow[r, "\text{instantiate } \underline{n}:=y"] \arrow[d, "\beta"] & (\lambda y.y)(x) \arrow[d, "\beta"'] \\
\underline{n} \arrow[r, "\text{instantiate } \underline{n}:=y"] & ??
\end{tikzcd}$$

The solution is to record the dependency of a meta-variable on other terms by writing $\underline{n}[y]$. An alternative solution is to delay substitutions by using *explicit* substitutions. Then we would have, e.g. $x[y:=t]=x$, for x a normal variable ($x\neq y$), but $\underline{n}[y:=t]\neq\underline{n}$ for a meta-variable. This approach is taken by [11] and [9]. We will follow the first approach, which is also taken by [13].
We illustrate the approach by redoing the same example above, but now with *dependencies* of meta-variables recorded.

$$\dfrac{?}{\exists f\forall x.f(x)=x} \qquad \dfrac{\dfrac{?}{\forall x.\underline{f}(x)=x}}{\exists f\forall x.f(x)=x}$$

$$\dfrac{\dfrac{\dfrac{?}{\forall x.(\lambda y.\underline{n}[y])(x)=x}}{\forall x.(\lambda y.\underline{n}[y])(x)=x}}{\exists f\forall x.f(x)=x} \qquad \dfrac{\dfrac{\dfrac{\dfrac{?}{\forall x.\underline{n}[x]=x}}{\forall x.(\lambda y.\underline{n}[y])(x)=x}}{\forall x.(\lambda y.\underline{n}[y])(x)=x}}{\exists f\forall x.f(x)=x}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\checkmark}{\forall x.x=x}}{\forall x.(\lambda y.y)(x)=x}}{\forall x.(\lambda y.y)(x)=x}}{\exists f\forall x.f(x)=x}$$

5. Using meta-variables to represent *unknown formulas*.

Suppose we are in arithmetics. The ‘usual’ induction principle is expressed by the formula $Ind_1 = \forall P{:}N\to\mathsf{Prop}.P(0) \wedge \forall n.P(n)\to P(n+1) \to \forall n.P(n)$. The ‘course-of-value’ induction principle is expressed by the formula $Ind_2 = \forall P.(\forall n(\forall k < n.P(k))\to P(n)) \to \forall n.P(n)$.

Suppose we want to prove that Ind_1 implies Ind_2. We will show how meta-variables can be used to prove this implication without having to make guesses ‘out of the blue’. After some obvious backward steps we have the initial open proof shown on the right ($\Phi(P)$ denotes $\forall n(\forall k < n.P(k))\to P(n))$.

$$\dfrac{\dfrac{\dfrac{\dfrac{Ind_1 \quad [\Phi(P)]^i}{?}}{\forall n.P(n)}}{\Phi(P)\to\forall n.P(n)}\,i}{Ind_2}$$

It is clear that we need to use the hypothesis Ind_1. To do that we have to eliminate the universal quantifier. Since we do not want to make guesses, we delay the choice and introduce a meta-variable $\underline{B}$ for the unknown predicate.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Ind_1}{\underline{B}(0) \wedge (\forall n.\underline{B}(n)\to\underline{B}(n+1))\to\forall n.\underline{B}(n)} \quad [\Phi(P)]^i}{?}}{\forall n.P(n)}}{\Phi(P)\to\forall n.P(n)}\,i}{Ind_2}$$

An obvious step towards solving the goal is to reduce it to these three subgoals:

$$(1)\ \dfrac{\dfrac{\Phi(P)}{?}}{\underline{B}(0)} \qquad (2)\ \dfrac{\dfrac{\Phi(P)}{?}}{\forall n.\underline{B}(n)\to\underline{B}(n+1)} \qquad (3)\ \dfrac{\dfrac{\Phi(P)}{?}}{\forall n.\underline{B}(n)\to P(n)}$$

The idea of course is to use (1) and (2) with implication elimination to obtain $\forall n.\underline{B}(n)$ from which using (3) we would derive $\forall n.P(n)$. To discard goal (3), it is sufficient to define $\underline{B}(n) := P(n) \wedge \underline{C}(n)$ where $\underline{C}(n)$ is a fresh meta-variable of type Prop. After the instantiation goals (1) and (2) look like this:

$$(1)\ \dfrac{\dfrac{\Phi(P)}{?}}{P(0) \wedge \underline{C}(0)} \qquad (2)\ \dfrac{\dfrac{\Phi(P)}{?}}{\forall n.(P(n) \wedge \underline{C}(n))\to(P(n+1) \wedge \underline{C}(n+1))}$$

Goal (2) is the hardest to solve. Without much creativity we observe that we can replace it by the following two goals: (2a) $P(n) \wedge \underline{C}(n) \to \underline{C}(n+1)$ and (2b) $\forall m.\underline{C}(m)\to P(m)$. Analysing goal (2b) shows that we are in the following situation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\Phi(P) : \forall n(\forall k < n.P(k))\to P(n)}{(\forall k < m.P(k))\to P(m)} \quad [\underline{C}(m)]^j}{?}}{P(m)}\,j}{\underline{C}(m)\to P(m)}$$

and it is now not difficult to see that $\underline{C}(n)$ can be taken to be the formula $\forall k < n.P(k)$ and the remaining goals (1) and (2a) are easily provable.

3 Open higher order predicate logic

We now give a formal definition of higher order predicate logic with open terms and open proofs, o-Predω. As usual, we first define the language, then the derivation rules and then the notion of derivability. We show that o-Predω is conservative over Predω, ordinary higher order predicate logic [3, 6]. This means that, if we have derived the higher order formula A in o-Predω without unfinished subproofs, then A is derivable in Predω.

Most of o-Predω is the same as Predω, but we present it nevertheless.

Definition 2 (Language of o-Predω).

- *The* domains*:* $\mathcal{D} ::= \mathsf{Prop} \mid \mathcal{B} \mid \mathcal{D}\to\mathcal{D}$*, where* Prop *is the domain of* propositions, $\mathcal{B}$ *is an arbitrary* base *domain. We use currying to represent domains of higher arity. Arbitrary domains will be denoted by* σ, τ.
- *The* terms*,* $Term(o\text{-}Pred\omega)$*:*
  - variables*, typed with a domain, notation* x_i^σ *or* $x_i : \sigma$.
  - application*:* $(f\,t) : \tau$*, if* $f : \sigma\to\tau$ *and* $t : \sigma$.
  - abstraction $(\lambda x{:}\sigma.q) : \sigma\to\tau$*, if* $q : \tau$.
  - formula constructors $A\wedge B : \mathsf{Prop}$, $A\to B : \mathsf{Prop}$, $A\vee B : \mathsf{Prop}$, $\neg A : \mathsf{Prop}$, $\forall x{:}\sigma.A : \mathsf{Prop}$, $\exists x{:}\sigma.A : \mathsf{Prop}$*, if* $A, B : \mathsf{Prop}$ *and* σ *a domain.*
  - meta-variable applications*:* $\underline{m}[t_1, \ldots, t_n] : \tau$*, if* $t_1 : \sigma_1, \ldots, t_n : \sigma_n$ *and* $\underline{m}[y_1 : \sigma_1, \ldots, y_n : \sigma_n] : \tau$ *is a* meta-variable.

Remark 3. We will call ’formula’ any term from the domain Prop. Note that the definition above allows also metavariables standing for formulas or functions producing formulas.

Remark 4. Meta-variables themselves are *not* terms. There are countably many meta-variables for every $\sigma_1, \ldots, \sigma_n, \tau$. We view the ‘assignemnt’ $[y_1 : \sigma_1, \ldots, y_n : \sigma_n] : \tau$ as being part of the meta-variable, so, for example $\underline{m}[y : \sigma] : \tau$ and $\underline{m}[y : \sigma] : \sigma$ are different meta-variables (but of course we will use different names as much as possible).

As terms with meta-variables are ordinary terms, meta-variables can occur in the arguments of another (or the same) meta-variable. For example, if $\underline{m}[y : \sigma, z : \sigma] : \sigma$ is a meta-variable and $f : \sigma\to\sigma$, then e.g. $\underline{m}[(f\,a), \underline{m}[a, (f\,a)]$ is a well-formed term.

If the domains that we quantify over are irrelevant, we will write $\forall x.A$ instead of $\forall x{:}\sigma.A$. Also, we will often write $\underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}] : \tau$ or just $\underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}]$ or $\underline{m}[\boldsymbol{y}]$ for $\underline{m}[y_1 : \sigma_1, \ldots, y_n : \sigma_n] : \tau$.

Definition 5 (Derivation Rules of o-Predω). *These are the same as for Predω plus an extra rule for representing unknown proofs. We show the rules for*

$\to$, $\forall$ and $\exists$, the conversion rule and the new rule (claim).

$$\dfrac{\begin{matrix}[A]^i\\ \vdots\\ B\end{matrix}}{A\to B}\,{}^{i}\ \to\text{-}I \qquad \dfrac{A\to B \quad A}{B}\ \to\text{-}E \qquad \dfrac{A[t/x]}{\exists x{:}\sigma.A}\ \exists\text{-}I^{**} \qquad \dfrac{\exists x{:}\sigma.A \quad \begin{matrix}[A]^i\\ \Sigma\\ B\end{matrix}}{B}\,{}^{i}\ \exists\text{-}E^{\dagger}$$

$$\dfrac{\begin{matrix}\Sigma\\ A\end{matrix}}{\forall x{:}\sigma.A}\ \forall\text{-}I^{*} \qquad \dfrac{\forall x{:}\sigma.A}{A[t/x]}\ \forall\text{-}E^{**} \qquad \dfrac{A}{B}\ (conv)^{\dagger\dagger} \qquad \dfrac{B_1,\dots,B_n}{A}\ (claim)$$

$*$: if $x \notin FV(assumptions\ of\ \Sigma)$ $\quad$ *$\dagger$: if $x \notin FV(assumptions\ of\ \Sigma \setminus \{A\})$*
$**$: for $t:\sigma$ $\quad$ *$\dagger\dagger$: if $A =_\beta B$*

The rule (claim) represents an unknown derivation of A from $B_1,\dots,B_n$. The hypotheses of the unknown derivation need to be specified explicitly, for example, because we need to check side conditions on assumptions in the rest of the rules (and these refer to the leaves of a derivation). This explicit representation of the hypotheses also allows us to represent the *forward steps* that one may want to do. Sometimes in derivations we will use the symbol '?' to denote the (claim) rule.

As usual, in the $\to$-I rule, the A-leaves that are labelled with i (notation $[A]^i$) are *discharged*, so they are no longer assumptions. Similarly, the A-leaves in the $\exists$-E rule are discharged.

Remark 6. In the conversion rule, $=_\beta$ is defined in terms of

$$(\lambda x{:}\sigma.t)q \longrightarrow_\beta t[q/x].$$

The *substitution* used here extends immediately to terms with *meta-variables*:

$$\underline{m}[t_1,\dots,t_n][q/x] := \underline{m}[t_1[q/x],\dots,t_n[q/x]]$$

We always work *modulo α-conversion*. Hence we adopt the *variable convention* (also 'Barendregt convention') that we always assume all *bound variables* (BV) to be *different* and *different* from the *free variables* (FV).

A *derivation tree* in o-Predω is the same as a derivation in Predω, except for the fact that we can now also have (claim) nodes in the tree. In the notion of *derivability* we also have to take the 'open parts' of the derivation tree (the (claim) nodes) into account. We will call these *goals*.

It is allowed that variables occur free in the goals. If a variable x occurs free in a specific formula in a derivation Σ, it may be *bound* in Σ (by a $\forall$-I rule or a $\exists$-E rule) or it may be *free* in Σ. We define these notions explicitly, as it is important for our interpretations of goals.

Definition 7 (Bound occurrences of variables in a derivation). *Given a derivation Σ and a formula A ocurring in Σ with $x \in FV(A)$. We say that* $x \in \mathrm{FV}(A)$ is bound in Σ *in one of the two following situations*

$$\dfrac{\begin{matrix}A\\ \vdots\\ B\end{matrix}}{\forall x{:}\sigma.B}\ \forall\text{-}I^{\dagger} \qquad\qquad \dfrac{\exists x{:}\sigma.C \quad \begin{matrix}[C]^i\\ \vdots\\ A\\ \vdots\\ B\end{matrix}}{B}\,{}^{i}\ \exists\text{-}E^{\dagger\dagger}$$

$\dagger$ with x free in all the formulas in the derivation between A and B (inclusive).

$\dagger\dagger$ with x free in all the formulas in the derivation between C and A (inclusive).

So, the notion of '$x \in \mathrm{FV}(A)$ is bound in Σ' is about a specific occurrence of A in the derivation Σ. It is defined by induction on Σ.

Definition 8 (Goals in a derivation).

1. *A* goal *in o-Predω is a judgement of the form*
$$x_1{:}\sigma_1,\dots,x_n{:}\sigma_n, A_1,\dots,A_n \rightsquigarrow B,$$
where $A_1,\dots,A_n,B$ are formulas and $x_1,\dots,x_n \in FV(A_1,\dots,A_n,B)$.
2. *A goal $x_1{:}\sigma_1,\dots,x_n{:}\sigma_n, A_1,\dots,A_n \rightsquigarrow B$, is a* goal of the derivation Σ *if Σ contains an application of the claim rule*
$$\dfrac{A_1 \ \dots\ A_n}{B}\,(claim)$$
with $x_1{:}\sigma_1,\dots,x_n{:}\sigma_n$ the free variables in $A_1,\dots,A_n,B$ that are bound in Σ.

Definition 9 (Derivability in o-Predω). *Given a set of formulas Γ, a set of goals G and a formula B, we say that B is* derivable *from $\Gamma; G$ in o-Predω, notation*

$$\Gamma; G \vdash B,$$

if there is a derivation Σ with conclusion B, (non-discharged) assumptions in Γ and all goals of Σ in G.

An important property of Predω is that the derivation rules are *compatible* with *substitution*. Hence derivations and derivability are compatible with substitution:

if $\Gamma \vdash A$ with derivation Σ, then $\Gamma[t/x] \vdash A[t/x]$ with derivation $\Sigma[t/x]$.

For o-Predω we have the same properties, where we have to take note that in a goal $x_1{:}\sigma_1,\dots,x_n{:}\sigma_n, A_1,\dots,A_n \rightsquigarrow B$, the variables $x_1,\dots,x_n$ are *bound* in $A_1,\dots,A_n,B$. Hence, we do not substitute for these variables.

Lemma 10 (Compatibility of derivability and substitution in o-Predω). *If $\Gamma; G \vdash_i A$, then $\Gamma[t/x]; G[t/x] \vdash_i A[t/x]$.*

Proof. By induction on the derivation tree Σ, one proves that, if Σ has conclusion A, assumptions Γ and goals G, then $\Sigma[t/x]$ is a well-formed derivation with conclusion $A[t/x]$, assumptions $\Gamma[t/x]$ and goals $G[t/x]$. $\square$

Example 11. Consider the following two derivations on the right, where in the first x occurs bound and in the second, x occurs free. The judgements associated with these two derivations are $A, C; (y{:}\sigma, A) \rightsquigarrow B(y), (y'{:}\sigma, C) \rightsquigarrow B(y')\to D(y') \vdash \forall x{:}\sigma.D(x)$ for the first and $A(x), C; A \rightsquigarrow B(x), C \rightsquigarrow B(x)\to D(x) \vdash D(x)$ for the second. Note what happens if we substitute t for x in the two derivations.

$$\dfrac{\dfrac{\dfrac{A}{B(x)}\,? \qquad \dfrac{C}{B(x)\to D(x)}\,?}{D(x)}}{\forall x{:}\sigma.D(x)}$$

$$\dfrac{\dfrac{A(x)}{B(x)}\,? \qquad \dfrac{C}{B(x)\to D(x)}\,?}{D(x)}$$

An important operation on derivations is *instantiation* (choosing a value for a meta-variable). Therefore, an equally important property for o-Predω is the compatibility of the derivation rules with *instantiation* of meta-variables. We first give a precise definition of instantiation.

Definition 12. *For $\underline{n}[\boldsymbol{y} : \boldsymbol{A}] : B$ a meta-variable and $t : B$ a term, we call*

$$\{\underline{n}[\boldsymbol{y} : \boldsymbol{A}] := t\}$$

an instantiation *(of $\underline{n}[\boldsymbol{y}]$). An instantiation extends immediately to all terms. The only interesting cases are the meta-variable applications. For readability, denote the instantiation $\{\underline{n}[\boldsymbol{y}] := t\}$ by *.*

$$(\underline{n}[\boldsymbol{q}])^* := t[\boldsymbol{q}^*/\boldsymbol{y}],$$
$$(\underline{m}[\boldsymbol{q}])^* := \underline{m}[\boldsymbol{q}^*] \text{ for } m, n \text{ different meta-variables.}$$

Note that the instantiations have to be applied heriditarily (also to $\boldsymbol{q}$ in the first case), because $\boldsymbol{q}$ may contain $\underline{n}$, so for example

$$\underline{n}[(f\,a), \underline{n}[a, (f\,a)]]\{n[x,y] := g\,x\,y\} \;=\; g\,(f\,a)(g\,(a\,(f\,a))).$$

The well-foundness of the instantiation can easily be proved by induction on the structure of the term in which we instantiate. Informally, we can think of the instantiation $M\{\underline{n}[\boldsymbol{y} : \boldsymbol{A}] := t\}$ as (a reduct of) $(\lambda n.M)(\lambda \boldsymbol{y}.t)$.

We sometimes have to rename bound variables in derivations before performing an instantiation. This problem is not really new for o-Predω, because it already appears in Predω (when performing a substitution). To make our point clear we treat the following example.

Example 13. Consider a derivation Σ of $(P\,\underline{n}[\;])$ and a derivation Θ of $(P\,\underline{n}[x])$, where Θ and Σ do not contain a free x in its assumptions. We can do a $\forall$-introduction and we can perform an instantiation, $\{\underline{n}[\;] := x + y\}$ on Σ, respectively $\{\underline{n}[x] := x + y\}$ on Θ. In the first derivation, to perform the instantiation, we first have to rename the bound variable x to z.

$$\dfrac{\dfrac{\Sigma}{(P\,\underline{n}[\;])}}{\forall x.(P\,\underline{n}[\;])} \overset{\{\underline{n}[\;]:=x+y\}}{\longmapsto} \dfrac{\dfrac{\Sigma^{\{n[\;]:=x+y\}}}{(P(x+y))}}{\forall z.(P(x+y)}$$

$$\dfrac{\dfrac{\Theta}{(P\,\underline{n}[x])}}{\forall x.(P\,\underline{n}[x])} \overset{\{\underline{n}[x]:=x+y\}}{\longmapsto} \dfrac{\dfrac{\Theta^{\{n[x]:=x+y\}}}{(P(x+y))}}{\forall x.(P(x+y))}$$

Lemma 14. *Instantiation is compatible with derivations in o-Predω: if $\Gamma; G \vdash_i A$ with derivation Σ, then $\Gamma^*; G^* \vdash_i A^*$ with derivation Σ^*, if we denote the instantiation by *.*

Proof. By induction on the structure of derivation trees. $\square$

Corollary 15. *o-Predω is conservative over Predω:*

If $\Gamma; \emptyset \vdash_i A$, then $\Gamma \vdash A$.

Proof. Suppose $\Gamma; \emptyset \vdash_i A$ with derivation Σ. This derivation may still contain meta-variables, say $\underline{n_1}, \ldots, \underline{n_k}$. Let $\{\underline{n_1}[\_] := x_1\}, \ldots, \{\underline{n_k}[\_] := x_k\}$ be instantiations for these meta-variables with fresh variables of appropriate sort. If we perform all these instantiations on Σ, we obtain a derivation Σ' of $\Gamma; \emptyset \vdash_i A$ and this derivation contains no more meta-variables. But then Σ' is also a derivation in Predω, because it contains no applications of the (claim) rule and all the terms occurring in it are Predω-terms. $\square$

Beyond open derivations

The logic o-Predω defined above gives us the answer to the problem of what an incomplete derivation is. Interactive theorem proving is however not only about individual derivations. Often we encounter situations where more advanced applications are needed:

1. **Proof reuse.** Consider example 2 in Section 2. There we had to prove the same formula twice because we needed it in two different places. One would probably want to avoid this unnecessary effort by *reusing* proofs that have already been done.
2. **'Scratch-paper' mechanism.** We may also wish to *explore* our knowledge to come to good instantiations, or to reject potential instantiations. For example, suppose we want to prove the formula $\exists x.\varphi(x) \wedge (x < 2)$ from $\forall x.\varphi(x)\to(0 < x)$ (see (1)). From the assumption and the formula that we want to prove we can derive some properties that $\underline{x}$ must have (2).

$$(1) \quad \dfrac{\dfrac{\forall x.\varphi(x)\to(0 < x)}{\varphi(\underline{x}) \wedge (\underline{x} < 2)}\,?}{\exists x.\varphi(x) \wedge (x < 2)}$$

From the conclusion of this extra derivation we may conclude that the only possible instantiation for $\underline{x}$ is $\{\underline{x} := 1\}$ (assuming the domain of x is the set of the natural numbers).

$$(2)\quad \frac{\dfrac{\varphi(\underline{x}) \wedge (\underline{x} < 2)}{\varphi(\underline{x})} \qquad \dfrac{\forall x.\varphi(x)\to(0 < x)}{\varphi(\underline{x})\to(0 < \underline{x})}}{\dfrac{(0 < \underline{x})}{(0 < \underline{x} < 2)}}$$

This simple example illustrates the need to sometimes pause the construction of the 'main' derivation, do some side computations or inferences *within* its scope and then come back with the results.

A general problem that emerges from the examples above is that open derivations do not (yet) capture the notion of *proof state*. The system o-Predω is just about individual open derivations. A proof state is, intuitively, a 'connected' set of derivations. We will use *type theory* to formalize the notion of proof state.

4 The Curry-Howard formulas-as-types embedding

The idea of the Curry-Howard formulas-as-types embedding is to map derivations of the logic, in our case Predω, to *proof terms* of an appropriate type theory, in our case λPredω. The type system λPredω has two 'sorts' or 'univereses': Type, representing the collection of all domains ($\mathcal{D}$ in the logic), and Prop, representing the collection of all formulas. (Hence $\mathsf{Prop} : \mathsf{Type}$, as the collection of formulas is iteself a domain). We do not give a definition of the type system λPredω but refer the reader to [6] or [?]. The type theory λPredω represents the logic Predω *faithfully*, because we have a *soundness* and a *completeness* result, stated as follows. (We use $\vdash_\lambda$ to denote derivability in the type theory and $\vdash_L$ to denote derivability in the logic.)

- **Soundness**: If $\Gamma \vdash_L A$ with derivation Σ, then $\Gamma_{\mathcal{L}}, \Gamma \vdash_\lambda [\![\Sigma]\!] : A$, where $\Gamma_{\mathcal{L}}$ declares the required parts of the language of Predω.
- **Completeness**: If $\Gamma \vdash_\lambda M : A$, then $\Gamma^- \vdash_L A$, where Γ^- selects the $A : \mathsf{Prop}$ for which $h{:}A \in \Gamma$.

For example the trivial derivation of $(Q\,x) \vdash_L (P\,x)\to(Q\,x)$ maps to

$$D{:}\mathsf{Type}, P, Q{:}D\to\mathsf{Prop}, x{:}D, h : (Q\,x) \vdash \lambda z{:}(P\,x).h : (P\,x)\to(Q\,x).$$

The formulas-as-types embedding can be extended to o-Predω if we define o-λPredω.

Definition 16. *The type system o-λPredω extends the type system λPredω allowing* meta-variable declarations *in the context of the form*

- $\underline{n}[\boldsymbol{y} : \boldsymbol{\sigma}] : \tau$ *with* $\boldsymbol{\sigma}, \tau : \mathsf{Type}$*,* open terms*,*
- $\underline{p}[\boldsymbol{y} : \boldsymbol{\sigma}, \boldsymbol{q} : \boldsymbol{A}] : B$ *with* $\boldsymbol{\sigma} : \mathsf{Type}$*,* $\boldsymbol{A}, B : \mathsf{Prop}$*,* open proofs*,*

The derivation rules are as follows.

$$\frac{\Gamma \vdash \boldsymbol{\sigma}{:}\mathsf{Type} \quad \Gamma \vdash \tau{:}\mathsf{Type}}{\Gamma, \underline{n}[\boldsymbol{y} : \boldsymbol{\sigma}] : \tau \vdash \mathsf{Ok}} \qquad \frac{\Gamma \vdash \boldsymbol{\sigma}{:}\mathsf{Type} \quad \Gamma, \boldsymbol{y} : \boldsymbol{\sigma} \vdash \boldsymbol{A} : \mathsf{Prop} \quad \Gamma, \boldsymbol{y} : \boldsymbol{\sigma} \vdash B : \mathsf{Prop}}{\Gamma, \underline{p}[\boldsymbol{y} : \boldsymbol{\sigma}, \boldsymbol{q} : \boldsymbol{A}] : B \vdash \mathsf{Ok}}$$

$$\frac{\Gamma \vdash \boldsymbol{t} : \boldsymbol{\sigma} \quad (\underline{n}[\boldsymbol{y} : \boldsymbol{\sigma}] : \tau) \in \Gamma}{\underline{n}[\boldsymbol{t}] : \tau} \qquad \frac{\Gamma \vdash \boldsymbol{t} : \boldsymbol{\sigma} \quad \Gamma \vdash \boldsymbol{r} : \boldsymbol{A}[\boldsymbol{t}/\boldsymbol{y}] \quad (\underline{p}[\boldsymbol{y} : \boldsymbol{\sigma}, \boldsymbol{q} : \boldsymbol{A}] : B) \in \Gamma}{\underline{p}[\boldsymbol{t}, \boldsymbol{r}] : B[\boldsymbol{t}/\boldsymbol{y}]}$$

$\Gamma \vdash \mathsf{Ok}$ *is the judgement that* Γ *is* well-formed.

The type system o-λPredω enjoys all the nice meta-theoretic properties, like Subject Reduction, Confluence and Strong Normalization. We do not give the precise statements and proofs here, because it is outside the scope of this paper.

Lemma 17. *The formulas-as-types embedding from Predω to λPredω extends to a sound and complete formulas-as-types embedding from o-Predω to o-λPredω.*

Proof. Given the derivation Σ of $\Gamma; G \vdash A$, the embedding is defined by induction on Σ. We show how $[\![\Sigma]\!]$ is defined for some cases. First we have to define the context in which $[\![\Sigma]\!]$ is well-typed: from $\Gamma = \{A_1, \ldots, A_n\}$, we construct $h_1{:}A_1, \ldots, h_n{:}A_n$, with $h_1, \ldots, h_n$ fresh variables. We denote this context also by Γ. A goal $(\boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{A}) \rightsquigarrow B$ is translated to the declaration $\underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{h}{:}\boldsymbol{A}] : B$, with $\underline{m}$ a fresh meta-variable. Thus the set of goals G is translated to a sequence of meta-variable declarations, which we also denote by G. Finally, we need a context to declare all the free variables and domain symbols that occur in Σ, Γ and G. This yields the context $\Gamma_{\mathcal{L}}$. To show that $[\![\Sigma]\!]$ is indeed a well-typed term of type A in $\Gamma_{\mathcal{L}}, \Gamma, G$ requires some meta-theory of the type system, which we do not provide here. In the following, if we write a derivation Σ with A on top and B below it, we mean that A and B are part of the derivation Σ.

1. If the last rule is (claim), then

$$\frac{\begin{matrix}\Sigma_1 \\ B_1\end{matrix} \quad \cdots \quad \begin{matrix}\Sigma_n \\ B_n\end{matrix}}{A}$$

   We construct $\Gamma_{\mathcal{L}}$ as the context of declarations for free variables and domains in Σ, Γ, G. For each Σ_i we construct Γ_i and G_i and by induction we find $[\![\Sigma_i]\!]$ such that $\Gamma_{\mathcal{L}}, \Gamma_i, G_i \vdash_i [\![\Sigma_i]\!] : B_i$. The goal is translated to a meta-variable $\underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{h}{:}\boldsymbol{B}] : A$, with $\boldsymbol{y}$ the variables bound in Σ. We define

$$[\![\Sigma]\!] := \underline{m}[\boldsymbol{y}, [\![\boldsymbol{\Sigma}]\!]]$$

   and find that $\Gamma_{\mathcal{L}}, \Gamma_1, G_1, \ldots, \Gamma_n, G_n, \underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{h}{:}\boldsymbol{B}] : A \vdash_i [\![\Sigma]\!] : B$.
2. If the last rule is ($\to$-I), then

$$\frac{\begin{matrix}[A]^i \ldots [A]^i \\ \Sigma_1 \\ B\end{matrix}}{A\to B}\, i$$

For Σ_1 we construct $\Gamma_{\mathcal{L}}$, Γ_1 and G_1 and by induction we find $[\![\Sigma_1]\!]$ such that $\Gamma_{\mathcal{L}}, \Gamma_1, G_1 \vdash_i [\![\Sigma_1]\!] : B$. The discharged occurrences of A correspond to variable declarations $h_1{:}A, \dots, h_n{:}A$ in Γ. We take $\Gamma := \Gamma \backslash (h_1{:}A, \dots, h_n{:}A)$ and $G := G_1$. We define

$$[\![\Sigma]\!] := \lambda h{:}A.([\![\Sigma_1]\!][h/h_1, \dots, h/h_n])$$

and find that $\Gamma_{\mathcal{L}}, \Gamma, G \vdash_i [\![\Sigma]\!] : A \to B$.

3. If the last rule is ($\forall$-I), then

$$\dfrac{\begin{array}{c}\Sigma_1\\ B\end{array}}{\forall x{:}\sigma.B}$$

For Σ_1 we construct $\Gamma_{\mathcal{L}}$, Γ_1 and G_1 and by induction we find $[\![\Sigma_1]\!]$ such that $\Gamma_{\mathcal{L}}, \Gamma_1, G_1 \vdash_i [\![\Sigma_1]\!] : B$. The quantified variable x may occur as a declaration in $\Gamma_{\mathcal{L}}$, but it does not occur free in Γ_1. So for Σ, we have $\Gamma_{\mathcal{L}} = \Gamma_{\mathcal{L}} \setminus (x{:}\sigma)$ and $\Gamma = \Gamma_1$. In the goals of Σ_1, x is free, whereas in the goals of Σ, x is bound. So, if $\underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{h}{:}\boldsymbol{C}] : A$ is a meta-variable declaration in G_1 with $x \in \mathrm{FV}(\boldsymbol{C}, A)$, then we replace this with the meta-variable declaration $\underline{m'}[x{:}\sigma, \boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{h}{:}\boldsymbol{C}] : A$ in G. We define

$$[\![\Sigma]\!] := \lambda x{:}\sigma.[\![\Sigma_1]\!]\{\underline{m}[\boldsymbol{y}, \boldsymbol{h}] := \underline{m'}[x, \boldsymbol{y}, \boldsymbol{h}]$$

and we find that $\Gamma_{\mathcal{L}} \setminus (x{:}\sigma), \Gamma, G \vdash [\![\Sigma]\!] : \forall x{:}\sigma.B$.

Proof states can now be represented as well-formed *contexts.* For *reuse* we also introduce *definitions* of (meta-)variables.

Definition 18. *The derivation rule for* definitions *is as follows:*

$$\dfrac{\Gamma, \boldsymbol{y} : \boldsymbol{A} \vdash q : B}{\Gamma, (\underline{n}[\boldsymbol{y} : \boldsymbol{A}] := q : B) \vdash \mathsf{Ok}} \quad \dfrac{\Gamma \vdash q : B}{\Gamma, (n := q : B) \vdash \mathsf{Ok}}$$

The *computation rules* for definitions are by *local* instantiation and *local* unfolding. That is because in general we do not want to instantiate all meta-variables at the same time (or unfold all definitions at the same time), but do that one by one. This reduction depends on the context Γ, where the definitions are recorded. If $(\underline{n}[\boldsymbol{y} : \boldsymbol{A}] := q : B) \in \Gamma$, resp. $(n := q : B) \in \Gamma$, the rule reads as follows.

$$t(\underline{n}[\boldsymbol{r}]) \xrightarrow{\Gamma}_{\delta} t(q[\boldsymbol{r}/\boldsymbol{y}])$$
$$t(n) \xrightarrow{\Gamma}_{\delta} t[q/n]$$

where $t(n)$ signifies one specific occurrence of n in t (and similarly for $t(\underline{n}[\boldsymbol{r}])$. Details of extensions of type theory with an explicit definition mechanism can be found in [12].

We illustrate how the type-theoretic contexts capture the notion of proof state by the following two examples.

Example 19. Consider the 'scratch-paper' example from Section 3. We can accomodate both the main derivation and the scratch derivation in one context. Let M be the term encoding the scratch derivation. The context now is follows.

$$\begin{array}{l}
\Gamma_0,\\
\underline{x}[] : \_,\\
h_{goal}[p : \forall x.\varphi(x) \to (0 < x)] : \varphi(\underline{x}) \wedge (\underline{x} < 2),\\
h_{scratch}[p : \forall x.\varphi(x) \to (0 < x)] := M(\underline{x}, p, h_{goal}) : (0 < \underline{x} < 2),\\
h_{main}[p : \forall x.\varphi(x) \to (0 < x)] := \langle \underline{x}, h_{goal}[p] \rangle : \exists x.\varphi(x) \wedge (\underline{x} < 2).
\end{array}$$

A tactic transforms proof states. As proof states are formalized as contexts, tactics should be context transformers. As an example we show the 'apply' tactic.

Example 20 (The **`Apply`** *tactic).* This tactic takes as inputs (1) a proof of a universally quantified or implicational formula U (2) a goal to be proved $(A_1, \dots, A_n) \rightsquigarrow B(s)$. It applies elimination rules to U, using optional arguments, until a proof of B is obtained or no elimination rule is applicable. In the latter case the tactic fails. If an optional argument is missing, then a new meta-variable is introduced and the elimination proceeds with it. Suppose $U = \forall x.C_1(x) \to \forall y.C_2(x, y) \to B(x)$.

$$\dfrac{\dfrac{A_1, \dots, A_n}{?}}{B(s)} \quad \xrightarrow{\mathit{Apply}\ \mathcal{D}\ s} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}}{\forall x.C_1(x) \to \forall y.C_2(x, y) \to B(x)}}{C_1(s) \to \forall y.C_2(s, y) \to B(s)} \quad \dfrac{\dfrac{A_1, \dots, A_n}{?}}{C_1(s)}}{\forall y.C_2(s, y) \to B(s)}}{C_2(s, \underline{y}[\,]) \to B(s)} \quad \dfrac{\dfrac{A_1, \dots, A_n}{?}}{C_2(s, \underline{y}[\,])}}{B(s)}$$

where $\mathcal{D}$ is some (open) derivation of $\forall x.C_1(x) \to \forall y.C_2(x, y) \to B(x)$. Note the introduction of the two new goals and the meta-variable $\underline{y}$. We can represent this tactic as a mapping between contexts:

$$\Gamma, \underline{h}[\boldsymbol{p} : \boldsymbol{A}] : B(s), \Delta \quad \xrightarrow{\mathit{Apply}\ M\ s} \quad \begin{array}{l}
\Gamma,\\
\underline{h'}[\boldsymbol{p} : \boldsymbol{A}] : C_1(s),\\
\underline{y}[\,] : \sigma,\\
\underline{h''}[\boldsymbol{p} : \boldsymbol{A}] : C_1(s, \underline{y}[\,]),\\
\underline{h}[\boldsymbol{p} : \boldsymbol{A}] := (M\ s\ \underline{h'}[\boldsymbol{p}]\ \underline{y}[\,]\ \underline{h''}[\boldsymbol{p}]) : B(s),\\
\Delta
\end{array}$$

where $\Gamma \vdash M : \forall x.C_1(x) \to \forall y.C_2(x, y) \to B(x)$ represents the derivation $\mathcal{D}$. Note the introduction and the use of the three new meta-variables $\underline{h'}$, $\underline{h''}$ and $\underline{y}$.

5 Related Work

Most of the work in the area of incomplete constructions is done in type theory where a number of systems of open terms in (dependently) typed λ-calculus exist

[13, 10, 4, 11, 9] (see [7] for an overview). They have evolved from existing typing systems (the Barendregt cube [3], ECC [8], Martin-Löf type theory, etc.) when their application in (interactive) theorem proving required formalization of the notion of incomplete term. Simply-typed systems have also been investigated but we they have only limited application in interactive theorem proving.

TypeLab [13] is based on ECC and represents unknown terms by meta-variables that are equipped with explicit substitutions. Each meta-variable is given a context and a type in that context and the idea is that the meta-variable stands for a well-typed term of the given type in that context.

OLEG [10] also takes ECC as a basis but the approach is to treat meta-variables declarations as part of the term. This is done by introducing special binders that locally declare meta-variables. In this way the position of the binder naturally expresses the context in which the meta-variable should be solved. Computations with terms containing meta-variable declarations are limited as such terms are not allowed to leak into types.

Bognar [4] generalizes the concept of context as used in the untyped λ-calculus [2] and introduces the $\lambda[\ ]$-cube. Along with the local declarations of meta-variables, these systems have explicit operators for instantiation.

6 Conclusions and Further Work

In this paper we formalized incomplete derivations in higher order predicate logic. By extending the Curry-Howard embeddings to incomplete proofs we fill in a gap that resulted from the focusing of the studies of incomplete objects exclusively to type theory.

Among the topics that need to be investigated is the question whether this framework is *flexible* enough to 'freely' do proofs in the way we like. This is a crucial point with respect to the practical applicability of interactive theorem proving . Related issues are the problems of finding cannonical set of *basic tactics* and *tacticals* that generate all (useful) tactics and the problems connected with viewing large proof states.

References

1. The system Mizar. http://www.mizar.org?
2. H. Barendregt. The λ-calculus: Its syntax and semantics, 1984.
3. Henk Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science*, pages 117–309. Clarendon Press, 1992.
4. Mirna Bognar. PhD thesis, VU Amsterdam, to appear, 2002.
5. Thierry Coquand and Gérard Huet. The calculus of constructions. In *Information and Computation*, number 76(2/3), pages 95–120. 1988.
6. J.H. Geuvers. *Logics and Type systems.* PhD thesis, University of Nijmegen, September 1993.
7. G.I. Jojgov. Systems for open terms: An overview. Technical Report CSR 01-03, Technische Universiteit Eindhoven, 2001.
8. Zhaohui Luo. *An Extended Calculus of Constructions.* PhD thesis, University of Edinburgh, July 1990.
9. Lena Magnusson. *The Implementation of ALF - a Proof Editor based on Martin-Löf Monomorphic Type Theory with Explicit Substitutions.* PhD thesis, Chalmers University of Technology / Göteborg University, 1995.
10. Conor McBride. *Dependently Typed Functional Programs and their Proofs.* PhD thesis, University of Edinburgh, 1999.
11. César A. Muñoz. *A Calculus of Substitutions for Incomplete-Proof Representation in Type Theory.* PhD thesis, INRIA, November 1997.
12. P. Severi and E. Poll. Pure Type Systems with definitions. In A. Nerode and Yu.V. Matiyasevich, editors, *Proceedings of LFCS'94, St. Petersburg, Russia*, number 813 in LNCS, pages 316–328, Berlin, 1994. Springer Verlag.
13. M. Strecker. *Construction and Deduction in Type Theories.* PhD thesis, Universität Ulm, 1998.