Proof Search with Set Variable Instantiation in the Calculus of Constructions*

Amy Felty

Bell Laboratories
Lucent Technologies, 700 Mountain Ave., Murray Hill, NJ 07974, USA
Amy.Felty@bell-labs.com

Abstract. We show how a procedure developed by Bledsoe for automatically finding substitution instances for set variables in higher-order logic can be adapted to provide increased automation in proof search in the Calculus of Constructions (CC). Bledsoe's procedure operates on an extension of first-order logic that allows existential quantification over set variables. The method finds *maximal solutions* for this special class of higher-order variables. This class of variables can also be identified in CC. The existence of a correspondence between higher-order logic and higher-order type theories such as CC is well-known. CC can be viewed as an extension of higher-order logic where the basic terms of the language, the simply-typed λ-terms, are replaced with terms containing dependent types. We adapt Bledsoe's procedure to the corresponding class of variables in CC and extend it to handle terms with dependent types.

1 Introduction

Both higher-order logic and higher-order type theories serve as the logical foundation of a variety of interactive tactic-style theorem provers. For example, both HOL [10] and Isabelle [16] implement higher-order logic, while Coq [6] implements the Calculus of Constructions (CC) type theory [5] and Nuprl [4] implements Martin-Löf type theory [14]. Much work has been carried out in both kinds of systems on building tactics and automating proof search. However, little work has been done on providing the means for exploiting proof search methods designed for one kind of system within the other. In this paper, we show how a particular proof search procedure designed for higher-order logic can be used to help automate the search for proofs in CC.

In some cases, such as the second-order polymorphic λ-calculus and second-order propositional logic, the correspondence between higher-order logic and higher-order type theories is exact and known as the Curry-Howard isomorphism [12]. Although it is less direct for CC, one way to view the correspondence was shown in Felty [9]. Intuitively, a functional type $P \rightarrow Q$ corresponds to an implication, while a dependent type $\forall x : P.Q$ corresponds to universal quantification. An important

* To appear in *Proceedings of the 13th International Conference on Automated Deduction, July 1996.*

difference is that while in CC the type P can be an arbitrary CC type, in higher-order logic (*e.g.*, Church's simple theory of types [3]) P must be a simple type. Although CC types include the types of the simply-typed λ-calculus, they also include much more.

Formally establishing such correspondences provides a framework in which to study how theorem proving techniques designed for one kind of system can be applied to proof search in the other. In this paper we show how the techniques described in Bledsoe [2] for the automatic discovery of substitutions for set variables can be incorporated directly into the search procedure for CC given by Dowek [7, 8]. In doing so, we both adapt these techniques to the type theoretic setting as well as extend them to handle the extra expressivity of dependent types. To incorporate dependent types, we consider not only single element membership such as $t \in A$, but also sets of tuples $\langle t_1, \ldots, t_n \rangle \in A$ where for $1 \leq i < j \leq n$, the type of t_j may depend on the type of t_i.

In Bledsoe [2], the procedure for finding substitution instances is implemented within an automatic theorem prover for natural deduction in first-order logic, thus extending it to handle existential quantification over a restricted set of second-order variables. The procedure has been successfully applied to obtain results in intermediate analysis, topology, logic, and program verification. To prove a theorem with set variables, the theorem prover makes two passes. The first finds maximal solutions for these variables. Once instantiated with the solutions, the formula becomes first-order, and the built-in strategy for proving first-order formulas is used. If the formula is provable, maximal solutions for set variables will lead to a proof. However, maximal solutions may be given during the first pass even though the formula is not provable. Thus the second pass is required. We take an example from Bledsoe [2] to illustrate maximal solutions. Consider the theorem

$$P(a) \supset \exists A(\forall x(x \in A \supset P(x)) \wedge \exists y(y \in A)).$$

A maximal solution for A is a term B that when substituted for A results in a provable formula, and such that for any other solution C, whenever $B \subseteq C$ it must be the case that C is the same as B. In this example, if we consider the two conjuncts separately, the set $\{x \mid P(x)\}$ is a maximal solution for A in the first, and the universal set is a solution for the second. Their intersection, $\{x \mid P(x)\}$, is a maximal solution for A in the formula as a whole. Note that there are often non-maximal solutions that result in provable formulas. In this case, for example, $\emptyset$ is a solution to the first conjunct. However, it is not a solution to the whole formula. Maximal solutions are more generally useful because solutions to subformulas are easily combined to obtain solutions to the whole formula.

Dowek's procedure for automatic proof search in CC is a complete procedure. It begins with the type representing the formula to be proved and attempts to find a term of that type representing a proof. However, although the procedure is complete, it is not efficient in practice because of the complexity of CC. In particular, the number of search paths quickly becomes prohibitive for most theorems. In the presence of assumptions with polymorphic types, for example, there may be infinite branching at many points during search. There are many ways to direct the search by tuning it to a particular class of theorems. Dowek proposes one that is incomplete but makes several restrictions including the elimination of infinite branching. This

procedure is still complete for many interesting sublanguages of CC such as the Logical Framework [11] or higher-order hereditary Harrop (hohh) formulas, which serve as the logical foundation for the λProlog logic programming language [15]. Completeness for full CC can be regained by applying the restricted procedure successively, proving a new lemma at each step and adding it to the assumptions before the next pass.

Our work can be viewed as the tuning of Dowek's procedure to find proofs more efficiently for theorems in the class considered by Bledsoe, *i.e.*, theorems in an extension of first-order logic with existential quantification over a certain class of higher-order variables. The procedure presented here does not eliminate any search paths, but instead adds some new ones that expand certain branches more quickly, in particular those that use maximal solutions for set variables. We can restrict this procedure to obtain a more practical procedure by eliminating some search paths of the original procedure as Dowek does, as well as by adding more fine-tuned control for better handling of our class of theorems. For example, we are implementing a version of Dowek's procedure that corresponds fairly directly to a one-pass version of Bledsoe's procedure. We use a goal-directed tactic style framework where each of the search primitives of the procedure is implemented as a tactic. These tactics can be combined to obtain a procedure that can prove most of the examples in Bledsoe [2] fully automatically. This specialized version could also be incorporated into Coq as a tactic, and used to automatically generate substitution instances when applied to goals of the appropriate form.

In the next section, we present CC and an extension of it, called Meta, developed by Dowek [7] and used as the foundation for his search procedure. In Sect. 3, we show how to map set theory into CC. We use the usual notion that a set is a predicate over elements of a particular type, or over other sets. We also define maximal solutions in our setting, which directly extend those in Bledsoe [2]. In Sect. 4, we present the search procedure, concentrating on our extensions to it. The complete procedure appears in Appendix A. Section 5 presents the theorems that justify the maximal solutions used in the search procedure. These theorems are direct extensions of the theorems in Bledsoe [2]. Finally, we conclude in Sect. 6.

2 The Calculus of Constructions

The syntax of terms of the Calculus of Constructions (CC) is given by the following grammar.

$$\mathit{Type} \mid \mathit{Prop} \mid x \mid PQ \mid \lambda x{:}P.Q \mid \forall x{:}P.Q$$

Here *Type* and *Prop* are constants, x is a syntactic variable ranging over variables, and P and Q are syntactic variables ranging over terms. We assume a denumerable set of CC variables. The variable x is bound in the expressions $\lambda x{:}P.Q$ and $\forall x{:}P.Q$. The former binding operator corresponds to the usual notion of λ-abstraction, while the latter corresponds to abstraction in dependent types. We write $P \to Q$ for $\forall x{:}P.Q$ when x does not occur in Q. In both kinds of bindings, we often leave off the type P when it can be easily inferred. A *context* is a set of pairs of the form $x{:}P$ where x is a variable and P a term.

Terms that differ only in the names of bound variables are identified. If x is a variable and P is a term then $[P/x]$ denotes the operation of substituting P for all free occurrences of x, systematically changing bound variables in order to avoid variable capture. The expression $[P_1/x_1, \ldots, P_n/x_n]$ denotes the simultaneous substitution of the terms $P_1, \ldots, P_n$ for distinct variables $x_1, \ldots, x_n$, respectively. The relation of convertibility up to α, β, and η is written as $=_{\beta\eta}$.

The rules of CC are given in Fig. 1. In these rules, s, s_1, and s_2, are either *Type* or *Prop*. In (INTRO), (PROD), and (ABS), we assume that the variable x does not already occur as the left hand side of a context item in Γ. We say that Γ is a *valid context* if there is a tree built using the rules of Fig. 1 such that $\vdash \Gamma$ context occurs at the root. We say that $\Gamma \vdash P : Q$ is *derivable* in CC if Γ is a valid context and this judgment occurs at the root of a tree built using the rules of Fig. 1. In this case, we also say that P *has type* Q or *is of type* Q in Γ, and that Q *is the type of* P in Γ. In addition, sometimes we simply write $\Gamma \vdash P : Q$ to indicate that this judgment is derivable. It will be clear from context when this is the case.

$$\vdash \langle\rangle \text{ context} \quad \text{(EMPTY-CTX)} \qquad \frac{\vdash \Gamma \text{ context} \qquad \Gamma \vdash P : s}{\vdash \Gamma, x{:}P \text{ context}} \text{(INTRO)}$$

$$\Gamma \vdash \mathit{Prop} : \mathit{Type} \quad \text{(PROP-TYPE)} \qquad \frac{x{:}P \in \Gamma}{\Gamma \vdash x : P} \text{(INIT)}$$

$$\frac{\Gamma \vdash P : s_1 \qquad \Gamma, x{:}P \vdash Q : s_2}{\Gamma \vdash \forall x{:}P.Q : s_2} \text{(PROD)}$$

$$\frac{\Gamma \vdash \forall x{:}R.Q : s \qquad \Gamma, x{:}R \vdash P : Q}{\Gamma \vdash \lambda x{:}R.P : \forall x{:}R.Q} \text{(ABS)}$$

$$\frac{\Gamma \vdash P_1 : \forall x{:}Q_1.Q_2 \qquad \Gamma \vdash P_2 : Q_1}{\Gamma \vdash P_1 P_2 : [P_2/x]Q_2} \text{(APP)}$$

$$\frac{\Gamma \vdash Q : s \quad \Gamma \vdash Q' : s \quad \Gamma \vdash P : Q \quad Q =_{\beta\eta} Q'}{\Gamma \vdash P : Q'} \text{(CONV)}$$

Fig. 1. CC Typing Rules

The search procedure in Sect. 4 operates on valid contexts in the slightly extended language called Meta [7]. The terms of Meta include all the terms of CC plus the additional constant *Extern*. Contexts in Meta also include existential quantification of the form $\exists x : P$ and equations between terms, written $P = Q$. The Meta typing rules include all those for CC plus the additional rules in Fig. 2. In addition, in the rules of Fig. 1, s_2 in (PROD) can be *Extern*, and s in (INTRO), (Q-INTRO), and (ABS) can also be *Extern*. Finally $=_{\beta\eta}$ in (CONV) is replaced by $=_{\beta\eta\Gamma}$ which denotes equality modulo $\beta\eta$-conversion plus the equations in Γ. In the rest of this paper, by context we mean Meta context unless otherwise stated.

Given a context Γ, a variable x is *universal in* Γ if there is a Q such that

$$\frac{\vdash \Gamma \text{ context} \qquad \Gamma \vdash P : s}{\vdash \Gamma, \exists x : P \text{ context}} \text{(Q-INTRO)} \qquad \frac{\Gamma \vdash P : Q \qquad \Gamma \vdash P' : Q}{\vdash \Gamma, P = P' \text{ context}} \text{(EQ-INTRO)}$$

$$\Gamma \vdash \mathit{Type} : \mathit{Extern} \quad \text{(TYPE-EXTERN)} \qquad \frac{\exists x : P \in \Gamma}{\Gamma \vdash x : P} \text{(Q-INIT)}$$

Fig. 2. Additional Typing Rules for Meta

$x : Q \in \Gamma$. The variable x is *existential in* Γ if $\exists x : Q \in \Gamma$. A term P is *closed in* Γ if every variable x occurring free in P is universal in Γ, and the type of x is closed in Γ.

We say that a term P is *atomic in context* Γ if there is a Q such that $\Gamma \vdash P : Q$ is derivable and there are terms $M_1, \ldots, M_n$, $n \geq 0$ such that $P =_{\beta\eta} xM_1 \ldots M_n$. If x is universal in Γ, we say that P is *rigid*. Otherwise, x is existential in Γ and we say that P is *flexible*. We say that K is a *base type in* Γ if $\Gamma \vdash K : \mathit{Type}$ is derivable and K is atomic in Γ.[2]

We equate terms on the right of a colon up to $\beta\eta$-convertibility. For example, we will often say "if term P has the form Q" to mean that P is $\beta\eta$-convertible to a term of the form Q. This convention is justified by rule (CONV).

3 Set Theory in the Calculus of Constructions

It is shown in Huet [13] that higher-order logic is contained within CC (and thus also in Meta). Terms are introduced that encode the connectives and it is shown that the corresponding natural deduction inference rules are provable in CC. Here, we use the abbreviations for the connectives, which are given in Fig. 3. For example, when we write the term $(\exists\ T\ \lambda x : T.A)$, it represents the Meta term $\forall C : \mathit{Prop}.((\forall x : T.A \rightarrow C) \rightarrow C)$, and encodes the formula $\exists_T x.A$ where $\exists_T$ is the existential quantifier at type T in higher-order logic. For readability, we will use infix notation for the binary connectives. As mentioned, implication and universal quantification are built into CC directly. Note that equality is Leibniz equality.

In set theory, from the fact that $a \in \{x : P(x)\}$, it is possible to immediately deduce $P(a)$. In our encoding in Meta, we build in this correspondence directly and define sets to be predicates of a certain class of types.

Definition 1. Term K is a *set type in context* Γ if $\Gamma \vdash K : \mathit{Type}$ is derivable and K has the form $\forall x_1 : A_1 \ldots \forall x_n : A_n.\mathit{Prop}$, where $n > 0$ and for $i = 1, \ldots, n$, A_i is a base type or set type in $\Gamma, x_1 : A_1, \ldots, x_{i-1} : A_{i-1}$. Term A is a *set in context* Γ if $\Gamma \vdash A : K$ and K is a set type in Γ.

[2] It is possible to include K such that $\Gamma \vdash K : \mathit{Prop}$ is derivable in the set of base types without any complication to the results in this paper. For ease of exposition, we choose not to.

$$
\begin{aligned}
\wedge &:= \lambda A, B : Prop.\forall C : Prop.((A \to B \to C) \to C) \\
\vee &:= \lambda A, B : Prop.\forall C : Prop.((A \to C) \to (B \to C) \to C) \\
\exists &:= \lambda T : Type.\lambda P : T \to Prop.\forall C : Prop.((\forall x : T.Px \to C) \to C) \\
\bot &:= \forall C : Prop.C \\
\top &:= \forall C : Prop.C \to C \\
\neg &:= \lambda A : Prop.A \to \bot \\
=_L &:= \lambda T : Prop.\lambda M, N : T.\forall P : T \to Prop.PM \to PN
\end{aligned}
$$

Fig. 3. CC Encoding of the Connectives of Higher-Order Logic

To illustrate, let Γ be the context $Nat : Type, 0 : Nat, s : Nat \to Nat$. Note that $Nat \to Prop$, $(Nat \to Prop) \to Prop$, $((Nat \to Prop) \to Prop) \to Prop$, *etc.* are all set types. Thus predicates over type Nat, predicates over sets of type Nat, predicates over sets of sets of type Nat, *etc.* are all sets. Figure 4 contains the abbreviations that we adopt for sets and set operations. The first abbreviation will always be used in a context where the types of $x_1, \ldots, x_n$ are known to be $A_1, \ldots, A_n$, respectively. We write $=_S$ for set equality.

$$
\begin{aligned}
\{\langle x_1, \ldots, x_n\rangle \mid A\} &:= \lambda x_1 : A_1 \ldots \lambda x_n : A_n.A \\
\langle M_1, \ldots, M_n\rangle \in B &:= (BM_1 \ldots M_n) \\
\emptyset &:= \lambda x_1 : A_1 \ldots \lambda x_n : A_n.\bot \\
B \subseteq C &:= \lambda x_1 : A_1 \ldots \lambda x_n : A_n.Bx_1 \ldots x_n \to Cx_1 \ldots x_n \\
B \cup C &:= \lambda x_1 : A_1 \ldots \lambda x_n : A_n.(Bx_1 \ldots x_n \vee Cx_1 \ldots x_n) \\
B \cap C &:= \lambda x_1 : A_1 \ldots \lambda x_n : A_n.(Bx_1 \ldots x_n \wedge Cx_1 \ldots x_n) \\
B =_S C &:= (B \subseteq C) \wedge (C \subseteq B)
\end{aligned}
$$

Provisos: $\lambda x_1 : A_1 \ldots \lambda x_n : A_n.A$, B, and C are sets in some context Γ
$\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash A : Prop$
$\Gamma \vdash B : \forall x_1 : A_1 \ldots \forall x_n : A_n.Prop$
$\Gamma \vdash C : \forall x_1 : A_1 \ldots \forall x_n : A_n.Prop$
$\Gamma \vdash M_i : [M_1/x_1, \ldots, M_{i-1}/x_{i-1}]A_i \quad$ for $i = 1, \ldots, n$

Fig. 4. CC Encoding of Sets

Returning to the example given in Sect. 1, we illustrate its proof within the framework of CC. Let Γ be the CC context $Nat : Type, P : Nat \to Prop, a : Nat$. Proving the theorem from Sect. 1 in higher-order logic corresponds to finding a CC term M such that the following judgment is derivable.

$$\Gamma \vdash M : Pa \to (\exists\ (Nat \to Prop)\ \lambda A.((\forall x : Nat.\langle x\rangle \in A \to Px) \wedge (\exists\ Nat\ \lambda y.\langle y\rangle \in A)))$$

Expanding the first $\exists$ and applying (ABS) three times in the backward direction, we get the following judgment as the rightmost premise. (We ignore the left premise of

each application. These are easily proved.)

$$\begin{array}{l}\Gamma, h_1 : Pa, C : Prop, \\ \quad h_2 : \forall A : Nat \rightarrow Prop.((\forall x : Nat.\langle x \rangle \in A \rightarrow Px) \wedge (\exists\ Nat\ \lambda y.\langle y \rangle \in A)) \rightarrow C \\ \qquad \vdash M' : C\end{array}$$

Here, M' is a new term such that M is equal to $\lambda h_1.\lambda C.\lambda h_2.M'$. Let Γ' be the context in the above judgment containing Γ, h_1, C, and h_2. The proof can be completed using two applications of (APP) from h_2, setting M' to $h_2 A M''$, where A and M'' are terms that must be filled in by proving the following two judgments.

$$\begin{array}{l}\Gamma' \vdash A : Nat \rightarrow Prop \\ \Gamma' \vdash M'' : (\forall x : Nat.\langle x \rangle \in A \rightarrow Px) \wedge (\exists\ Nat\ \lambda y.\langle y \rangle \in A)\end{array}$$

As in Sect. 1, we take A to be $\{x \mid Px\}$, which by definition is just $\lambda x : Nat.Px$ which is η-equivalent to P. The first judgment is directly provable, and the second becomes

$$\Gamma' \vdash M'' : (\forall x : Nat.\langle x \rangle \in P \rightarrow Px) \wedge (\exists\ Nat\ \lambda y.\langle y \rangle \in P)$$

which, after expanding definitions, is also directly provable.

A term A is said to *occur positively* in a term x, PQ, or $\lambda x : P.Q$ if A occurs in any of these terms. Term A *occurs positively* (*negatively*) in $\forall x : P.Q$ or $P \rightarrow Q$ if A occurs positively (negatively) in Q or negatively (positively) in P. Figure 5 shows maximal solutions for variable A or B in various subformulas. All of these subformulas are assumed to occur positively in the theorem to be proved. A is assumed to occur only in the form $\langle M_1, \ldots, M_n \rangle \in A$, and similarly for B. These are the solutions considered by Bledsoe that are handled by our version of Dowek's procedure. As stated, our solutions are generalizations of Bledsoe's solutions in that they allow tuples instead of singleton members of sets and dependencies may occur in the types of the tuples.

We will use these rules directly in the procedure in the next section. They are justified to some degree by the theorems in Sect. 5. The first rule is the one that was used to determine the solution of the first conjunct of the example above. Although the second rule looks complicated, it is just the dependent-type version of solving for $fx \in B \rightarrow P'(x)$ obtaining maximal solution $\{z \mid \forall x(z = fx \rightarrow P'(x))\}$. In the CC version, the types of the last m arguments of the tuple can depend on the types of the first j arguments but not on the types of each other. The remaining rules are fairly straightforward.

4 Proof Search with Set Variable Instantiation

Dowek's search procedure can be described as a set of *search operations* that transform one Meta context to another, ending when one with no existential variables or unsolved equations is reached. Our extension adds one new operation, called SETVAR, which instantiates set variables with maximal solutions. A search path is elaborated by starting with a context, non-deterministically choosing one operation that applies, and repeating until no more operations apply. For a complete procedure, control must be added in such a way that all branches get explored. Simple

Subformula	Solution for A or B
1. $\langle x_1,\ldots,x_n\rangle \in A \to P x_1 \ldots x_n$	$\longrightarrow \{\langle x_1,\ldots,x_n\rangle \mid P x_1 \ldots x_n\}$
2. $\langle x_1,\ldots,x_j, f_1 x_1 \ldots x_n,\ldots, f_m x_1 \ldots x_n\rangle \in B \to P' x_1 \ldots x_n$	$\longrightarrow \{\langle x_1,\ldots,x_j,z_1,\ldots,z_m\rangle \mid$ $\forall x_{j+1}:B_{j+1}\ldots\forall x_n:B_n.z_1 =_L f_1 x_1 \ldots x_n$ $\to\cdots\to z_m =_L f_m x_1 \ldots x_n \to P' x_1 \ldots x_n\}$
3. $\langle M_1,\ldots,M_n\rangle \in A \to Q$	$\longrightarrow \{\langle x_1,\ldots,x_n\rangle \mid x_1 =_L M_1 \to\cdots\to x_n =_L M_n \to Q\}$
4. $\neg(\langle M_1,\ldots,M_n\rangle \in A)$	$\longrightarrow \{\langle x_1,\ldots,x_n\rangle \mid \neg(x_1 =_L M_1 \wedge\cdots\wedge x_n =_L M_n)\}$
5. Q ($\langle M_1,\ldots,M_n\rangle \in A$ occurs positively or not at all)	$\longrightarrow \{\langle x_1,\ldots,x_n\rangle \mid \top\}$
6. If 1-4 yield $\{\langle y_1,\ldots,y_q\rangle \mid Q' y_1 \ldots y_q\}$, and s is a free variable of type T in Q'	$\longrightarrow \{\langle y_1,\ldots,y_q\rangle \mid (\exists\ T\ \lambda s.Q' y_1 \ldots y_q)\}$

Provisos: A and B are sets in some context Γ.
$A, B, x_1,\ldots,x_n, z_1,\ldots,z_m$ do not occur free in $P, P', Q, f_1,\ldots,f_m, M_1,\ldots,M_n$.
$x_1,\ldots,x_n, s$ do not occur elsewhere in the surrounding formula or context.
$x_1,\ldots,x_n$ are either bound by universal quantification or are universal in the surrounding context.
$\Gamma \vdash A : \forall x_1:A_1\ldots\forall x_n:A_n.Prop$
$\Gamma \vdash B : \forall x_1:A_1\ldots\forall x_j:A_j.A_{j+1}\to\cdots\to A_{j+m}\to Prop$
$\Gamma \vdash P : \forall x_1:A_1\ldots\forall x_n:A_n.Prop$
$\Gamma \vdash P' : \forall x_1:A_1\ldots\forall x_j:A_j.\forall x_{j+1}:B_{j+1}\ldots\forall x_n:B_n.Prop$
$\Gamma \vdash Q : Prop$
$\Gamma \vdash f_i : \forall x_1:A_1\ldots\forall x_j:A_j.\forall x_{j+1}:B_{j+1}\ldots\forall x_n:B_n.A_{j+i}$ for $i = 1,\ldots,m$
$\Gamma \vdash M_i : [M_1/x_1,\ldots,M_{i-1}/x_{i-1}]A_i$ for $i = 1,\ldots,n$

Fig. 5. Maximal Solutions for Various Subformulas

breadth-first search is not enough since some nodes have infinitely many branches. Although such control can be added, we do not discuss the details here. For our extension to be effective, SETVAR must be given priority in the presence of contexts containing set variables.

In this section, we present only the SETVAR operation and the BACKCHAIN operation, which corresponds to the usual notion of backchaining in theorem proving or logic programming. The remaining operations of the complete procedure are given in Appendix A.

Search begins with a valid Meta context, usually of the form $\Gamma, \exists h:Q$ where Γ contains only universally quantified variables, Q is the theorem to be proved, and h is the variable to be instantiated with a proof, if there is one. Set variables usually arise from existential quantification over variables in Q having set types. Via the step by step application of operations, existential quantifiers result in the addition of new existential variables to the context, which may eventually get instantiated with closed terms representing sets.

Several concepts from Dowek [7] are needed for the presentation of the operations, including the notion of context substitution. Let σ be a set of tuples of the

form $\langle x, \Delta, M\rangle$ where x is a variable, Δ is a context containing only existential variables and equations, and M is a term. The set σ is a *substitution* if for any variable x, there is at most one tuple in σ with x as its first component. Let Γ be a context. Then σ *is a valid substitution in* Γ if for every tuple $\langle x, \Delta, M\rangle$ in σ, the context Γ, Δ is valid and M and x have the same type in Γ. Such substitutions will be applied to both contexts and terms. To apply a substitution σ to a term M, denoted σM, we consider the set of pairs obtained from σ by ignoring the middle argument. Substitution is then the usual notion of replacing variables with the corresponding terms renaming bound variables when necessary to avoid variable capture. The application of substitution σ to a context Γ, denoted $\sigma\Gamma$, is defined recursively as follows.

- If Γ is $\langle\rangle$, $\sigma\Gamma$ is $\langle\rangle$.
- If Γ is $\Gamma', x:T$, then $\sigma\Gamma$ is $\sigma\Gamma', x:\sigma T$.
- If Γ is $\Gamma', \exists x:T$, then if there is a tuple $\langle x, \Delta, M\rangle$ in σ, $\sigma\Gamma$ is $\sigma\Gamma', \Delta$. Otherwise, $\sigma\Gamma$ is $\sigma\Gamma', \exists x:\sigma T$.
- If Γ is $\Gamma', M = N$, then $\sigma\Gamma$ is $\sigma\Gamma', \sigma M = \sigma N$.

A valid context Γ is a *success context* if it contains no existential variables and all its equations relate $\beta\eta$-convertible terms. A valid context Γ is a *failure context* if it contains an equation that relates two terms that have no free occurrences of existential variables and that are not $\beta\eta$-convertible. Let Γ be a valid Meta context. A *candidate variable* is an existential variable $\exists z:T$ such that T has the form $\forall x_1 : A_1 \ldots \forall x_n : A_n . x M_1 \ldots M_m$ where $n, m \geq 0$ and x is rigid in $\Gamma, x_1 : A_1, \ldots, x_n : A_n$. It is shown in Dowek [7] that during search at least one such existential variable always exists, and that if no rule applies to any such existential variable in Γ, then Γ is a success or failure context.

Definition 2 SETVAR **operation.** Let Γ be a valid Meta context and $\exists z : T$ a candidate variable in Γ. In order for this operation to apply, T must have the form $\forall x_1 : A_1 \ldots \forall x_n : A_n . \mathit{Prop}$, where for some r such that $0 < r \leq n$ $\forall x_r : A_r \ldots \forall x_n : A_n . \mathit{Prop}$ is a set type. Also, z must occur no more than once in the type of any universal or existential variable in Γ, must only occur in closed types with outermost universal quantifiers $\forall x_1 : A_1 \ldots \forall x_{r-1} : A_{r-1}$, and must always occur in the form $\langle N_r, \ldots, N_n\rangle \in z x_1 \ldots x_{r-1}$. Let $P_1, \ldots, P_q$ be the terms in Γ in which z appears. For $k = 1, \ldots, q$, one of 1, 2, 3, or 4 below must hold:

1. All of the following hold:
   (a) P_i has the form $\forall y_1 : Q_1 \ldots \forall y_k : Q_k . P_i'$ where $k \geq n$ and P_i' is atomic.
   (b) For $i = r, \ldots, n$, there is a j such that $1 \leq j \leq k$ and y_j is x_i and Q_j is A_i.
   (c) For $j = 1, \ldots, k$, if there is no i, such that $r \leq i \leq n$ and y_j is x_i, then y_j does not appear free in P_i'.

   and one of the following holds:
   (a) P_i' has the form of subformula 1 in Fig. 5 such that the appropriate provisos hold. Then P_i' is $\langle x_r, \ldots, x_n\rangle \in z x_1 \ldots x_{r-1} \rightarrow P x_r \ldots x_n$.
   (b) P_i' has the form of subformula 2 in Fig. 5 such that the appropriate provisos hold. Then P_i' is $\langle x_r, \ldots, x_j, f_1 x_r \ldots x_p, \ldots, f_m x_r \ldots x_p\rangle \in z x_1 \ldots x_{r-1} \rightarrow P' x_r \ldots x_p$.

2. P_i' has the form of subformula 3 in Fig. 5 such that the appropriate provisos hold. Then P_i' is $\langle N_r, \ldots, N_n \rangle \in z x_1 \ldots x_{r-1} \rightarrow Q$.
3. P_i' has the form of subformula 4 in Fig. 5 such that the appropriate provisos hold. Then P_i' is $\neg(\langle N_r, \ldots, N_n \rangle \in z x_1 \ldots x_{r-1})$.
4. Variable z occurs positively in P_i.

Then for $i = 1, \ldots, q$, view $z x_1 \ldots x_{r-1}$ as a single set variable and let Q_i be the solution for $z x_1 \ldots x_{r-1}$ in P_i' according to rules 1-5 of Fig. 5. If appropriate, apply rule 6 of the figure as many times as possible to Q_i to obtain Q_i'. Let Q be the term $Q_1' \cap \cdots \cap Q_q'$. Let σ be the singleton set containing the tuple $\langle z, \langle\rangle, \lambda x_1 : A_1 \ldots \lambda x_{r-1} : A_{r-1}.Q \rangle$. SETVAR is the operation that replaces Γ with $\sigma\Gamma$.

Definition 3 BACKCHAIN **operation.** Let Γ be a valid Meta context and $\exists z : T$ a candidate variable in Γ. T has the form $\forall x_1 : A_1 \ldots \forall x_n : A_n . x M_1 \ldots M_m$ where $n, m \geq 0$ and x is rigid in $\Gamma, x_1 : A_1, \ldots, x_n : A_n$. If there is a universal variable $w : Q'$ to the left of z in Γ or in $x_1 : A_1, \ldots, x_n : A_n$, such that Q' has the form $\forall y_1 : Q_1' \ldots \forall y_q : Q_q' . y N_1 \ldots N_p$ and y is x or any existential variable in Γ, then we can "backchain on" Q' as follows. Define

$$\begin{array}{l} Q_1 := Q_1', \quad Q_2 := \lambda y_1 : Q_1' . Q_2', \quad \ldots, \quad Q_q := \lambda y_1 : Q_1' \ldots \lambda y_{q-1} : Q_{q-1}' . Q_q' \\ Q := \lambda y_1 : Q_1' \ldots \lambda y_q : Q_q' . y N_1 \ldots N_p. \end{array}$$

$$\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash w : \forall y_1 : Q_1 . \forall y_2 : (Q_2 y_1) \ldots \forall y_q : (Q_q y_1 \ldots y_{q-1}) . Q y_1 \ldots y_q$$

is thus derivable. Let Γ_1 be the context

$$\begin{array}{l} \exists h_1 : \forall x_1 : A_1 \ldots \forall x_n : A_n . Q_1, \exists h_2 : \forall x_1 : A_1 \ldots \forall x_n : A_n . Q_2 (h_1 x_1 \ldots x_n), \ldots, \\ \exists h_q : \forall x_1 : A_1 \ldots \forall x_n : A_n . Q_q (h_1 x_1 \ldots x_n) \ldots (h_{q-1} x_1 \ldots x_n) \end{array}$$

and Γ_2 the context containing the equation

$$\forall x_1 : A_1 \ldots \forall x_n : A_n . Q (h_1 x_1 \ldots x_n) \ldots (h_q x_1 \ldots x_n) = \forall x_1 : A_1 \ldots \forall x_n : A_n . x M_1 \ldots M_m.$$

Let σ be $\{\langle z, (\Gamma_1, \Gamma_2), \lambda x_1 : A_1 \ldots \lambda x_n : A_n . w (h_1 x_1 \ldots x_n) \ldots (h_q x_1 \ldots x_n) \rangle\}$. The BACKCHAIN operation replaces Γ with $\sigma\Gamma$.

The q existential variables in Γ' are the subgoals obtained by backchaining. Note that here, backchain takes place "under" a binder of universal quantifiers.

The SETVAR and BACKCHAIN are sufficient for proving the example given in Sect. 1 as well as most of the examples in Bledsoe [2]. To illustrate, we return again to our example. Again, let Γ be the context Nat: *Type*, $P : Nat \rightarrow Prop, a : Nat$. We begin with the following Meta context.

$$\Gamma, \exists M : P a \rightarrow (\exists\ (Nat \rightarrow Prop)\ \lambda A . ((\forall x : Nat . \langle x \rangle \in A \rightarrow P x) \wedge (\exists\ Nat\ \lambda y . \langle y \rangle \in A)))$$

Expanding the first $\exists$, we get

$$\begin{array}{l} \Gamma, \exists M : P a \rightarrow \forall C : Prop. \\ \quad (\forall A : Nat \rightarrow Prop . ((\forall x : Nat . \langle x \rangle \in A \rightarrow P x) \wedge (\exists\ Nat\ \lambda y . \langle y \rangle \in A)) \rightarrow C) \rightarrow C. \end{array}$$

The only candidate variable is M. BACKCHAIN is applied with Q' as

$$\forall A : Nat \rightarrow Prop . ((\forall x : Nat . \langle x \rangle \in A \rightarrow P x) \wedge (\exists\ Nat\ \lambda y . \langle y \rangle \in A)) \rightarrow C$$

and Γ' as

$$\begin{array}{l}\exists A' : Pa \rightarrow \forall C : Prop.Q' \rightarrow Nat \rightarrow Prop, \\ \exists M' : \forall h_1 : Pa.\forall C : Prop.\forall h_2 : Q'. \\ \quad ((\forall x : Nat.\langle x\rangle \in A' h_1 C h_2 \rightarrow Px) \wedge (\exists\ Nat\ \lambda y.\langle y\rangle \in A' h_1 C h_2))\end{array}$$

resulting in the substitution with the single tuple

$$\langle M, \Gamma', \lambda h_1 : Pa.\lambda C : Prop.\lambda h_2 : Q'.h_2(A' h_1 C h_2)(M' h_1 C h_2)\rangle.$$

(We leave out the equations in Γ' since they equate closed equivalent terms.) Applying this substitution, we obtain Γ, Γ'. We cannot yet apply SETVAR because the type of M' has two occurrences of $A' h_1 C h_2$. Although we do not give the details, after expanding $\wedge$, BACKCHAIN with z as M' can be applied to obtain the following context where Q'' is $(\forall x : Nat.\langle x\rangle \in A' h_1 C h_2 \rightarrow Px) \rightarrow (\exists\ Nat\ \lambda y.\langle y\rangle \in A' h_1 C h_2) \rightarrow C'$.

$$\begin{array}{l}\Gamma, \exists A' : Pa \rightarrow \forall C : Prop.Q' \rightarrow Nat \rightarrow Prop, \\ \exists M_1 : \forall h_1 : Pa.\forall C : Prop.\forall h_2 : Q'.\forall C' : Prop.Q'' \rightarrow (\forall x : Nat.\langle x\rangle \in A' h_1 C h_2 \rightarrow Px), \\ \exists M_2 : \forall h_1 : Pa.\forall C : Prop.\forall h_2 : Q'.\forall C' : Prop.Q'' \rightarrow (\exists\ Nat\ \lambda y.\langle y\rangle \in A' h_1 C h_2)\end{array}$$

It is now possible to apply SETVAR with the type of M_1 above as P_1 in the definition of SETVAR, and the type of M_2 as P_2. Clause 1 of SETVAR holds for P_1 while clause 4 holds for P_2. Applying the rules of Fig. 5, we get $Q'_1 := \lambda h_1 : Pa.\lambda C : Prop.\lambda h_2 : Q'.\{\langle x\rangle \mid Px\}$ and $Q'_2 := \lambda h_1 : Pa.\lambda C : Prop.\lambda h_2 : Q'.\{\langle x\rangle \mid \top\}$. We obtain the substitution $\langle A', \langle\rangle, \lambda h_1 : Pa.\lambda C : Prop.\lambda h_2 : Q'.Q'_1 \cap Q'_2\rangle$. The resulting context after substitution is

$$\begin{array}{l}\Gamma, \exists M_1 : Pa \rightarrow \forall C : Prop.Q' \rightarrow \forall C' : Prop.Q'' \rightarrow (\forall x : Nat.\langle x\rangle \in Q'_1 \cap Q'_2 \rightarrow Px), \\ \exists M_2 : Pa \rightarrow \forall C : Prop.Q' \rightarrow \forall C' : Prop.Q'' \rightarrow (\exists\ Nat\ \lambda y.\langle y\rangle \in Q'_1 \cap Q'_2)\end{array}$$

which is easily transformed to a success context via several applications of BACKCHAIN.

Several similarities and differences between Bledsoe's procedure and ours are worth noting. First, the restriction in SETVAR requiring z to occur no more than once in any context item is also required by Bledsoe. In both cases, a series of applications of other operations will often reduce the formula (Meta context in our case) to this form. Second, Bledsoe's procedure uses Skolemization to eliminate universal quantifiers. Here, we retain the universal quantifiers as Dowek does and add the appropriate provisos on variable occurrences (see Fig. 5). Third, note that in the example above, as operations are applied, the binder of universal quantifications of newly added existential variables is always the same or an extension of previous ones. One consequence of keeping these binders from step to step during search is that the notion of set variable must actually be extended to allow applications of an existential variable to zero or more universally quantified variables. This more general notion of set variable was easily incorporated into the SETVAR operation. Fourth, Bledsoe's procedure solves for maximal solutions across subformulas joined by conjunction in much the same way we do, by taking the intersection of the maximal solutions for each individual conjunct. (Here, conjuncts must first be reduced as in the example above so that each occurrence of a set variable occurs in a different context item.) However, disjunction in the two procedures is handled differently. Bledsoe's procedure keeps track of a set of possible maximal solutions, one for each disjunct. Here, the different solutions are spread across different search paths. A

context of the form $\Gamma, \exists h : \forall x_1 : A_1 \ldots \forall x_n : A_n.(A \vee B), \Gamma'$ can be reduced in least two ways using BACKCHAIN to a context of either of the following two forms:

$$\Gamma, \exists h_1 : \forall x_1 : A_1 \ldots \forall x_n : A_n.A, \Gamma' \quad \text{or} \quad \Gamma, \exists h_2 : \forall x_1 : A_1 \ldots \forall x_n : A_n.B, \Gamma'.$$

This approach fits more naturally within the context of Dowek's procedure. In addition, it simplifies bookkeeping tasks involved in keeping track of multiple solutions.

We say that valid Meta context Γ *has a derivation* if applying the search operations leads to a success context. We extend the following result from Dowek [7] to express soundness and completeness of the search procedure with SETVAR.

Theorem 4. *Let Γ be a CC context and P a term such that $\Gamma, \exists x : P$ is a valid Meta context. If $\Gamma, \exists x : P$ has a derivation, then there exists a term h such that $\Gamma \vdash h : P$ is derivable in CC. Conversely, if there exists a term h such that $\Gamma \vdash h : P$ is derivable in CC, then $\Gamma, \exists x : P$ has a derivation,*

The converse follows directly since SETVAR only adds an operation. Soundness is extended to our procedure by showing that SETVAR preserves derivability in CC. We do not present the details of this proof. It involves a simple extension of several lemmas in Dowek [7], mainly showing that the substitutions arising from SETVAR are valid. The proof of the above theorem then follows directly.

5 Maximal Solutions for Set Variables

Definition 5. Let Γ be a valid context and K, B terms such that K is a set type, B is a set, and $\Gamma \vdash B : K$ is derivable. Let Γ' be a valid context of the form $\Gamma, \exists A : K, \Gamma''$. Let σ be the substitution containing the single tuple $\langle A, \langle\rangle, B\rangle$. B is a *maximal solution for A in Γ'* if $\sigma\Gamma'$ has a derivation and for any C such that the following hold:

1. $\Gamma \vdash C : K$ is derivable.
2. $\sigma'\Gamma'$ has a derivation, where σ' is the substitution $\{\langle A, \langle\rangle, C\rangle\}$.
3. There is a term M such that $\Gamma \vdash M : B \subseteq C$ is derivable.

then there is a term N such that $\Gamma \vdash N : B =_S C$ is derivable.

Theorems 6-9 justify the maximal solutions given in Fig. 5, while Theorem 10 justifies taking the intersection of maximal solutions of different occurrences of a set variable as done in SETVAR in Sect. 4. We do not give the details of the proofs here. They are straightforward extensions of the proofs in Bledsoe [2].

Theorem 6. *Let Γ be a valid Meta context such that Γ has a derivation. Let K be a set type in Γ such that K has the form $\forall x_1 : A_1 \ldots \forall x_n : A_n.Prop$ for some $n > 0$. Let P be a term such that P is closed in Γ, $Px_1 \ldots x_n$ is atomic in $\Gamma, x_1 : A_1, \ldots, x_n : A_n$, and $\Gamma \vdash P : K$ is derivable. Then $\{\langle x_1, \ldots, x_n\rangle \mid Px_1 \ldots x_n\}$ is a maximal solution for A in $\Gamma, \exists A : K, \exists h : \forall x_1 : A_1 \ldots \forall x_n : A_n.\langle x_1 \ldots x_n\rangle \in A \rightarrow Px_1 \ldots x_n$.*

Note that it is easy to extend this theorem so that the outermost universal quantifiers in the type of h and A may contain additional quantified variables $y_1, \ldots, y_m$, as long as the variables don't occur free in P, and $Ay_1 \ldots y_m$ is viewed as a single set variable. This more general form is the one actually used in SETVAR. Similar generalizations can be made for all of the theorems below.

Theorem 7. *Let Γ be a valid Meta context such that Γ has a derivation. Let K be a set type in Γ such that K has the form $\forall x_1 : A_1 \ldots \forall x_j : A_j . A_{j+1} \rightarrow \cdots \rightarrow A_{j+m} \rightarrow Prop$ for some $j \geq 0$ and $m > 0$. For $n \geq j$, let $P', f_1, \ldots, f_m, B_{j+1}, \ldots, B_n$ be closed terms such that the following are derivable*

$$\begin{array}{l} \Gamma \vdash P' : \forall x_1 : A_1 \ldots \forall x_j : A_j . \forall x_{j+1} : B_{j+1} \ldots \forall x_n : B_n . Prop \\ \Gamma \vdash f_i : \forall x_1 : A_1 \ldots \forall x_j : A_j . \forall x_{j+1} : B_{j+1} \ldots \forall x_n : B_n . A_{j+i} \quad \text{for } i = 1, \ldots, m \end{array}$$

and $P' x_1 \ldots x_n$ is atomic in $\Gamma, x_1 : A_1, \ldots, x_j : A_j, x_{j+1} : B_{j+1}, \ldots, x_n : B_n$. Then $\{\langle x_1, \ldots, x_j, z_1, \ldots, z_m \rangle \mid \forall x_{j+1} : B_{j+1} \ldots \forall x_n : B_n . z_1 =_L f_1 x_1 \ldots x_n \rightarrow \cdots \rightarrow z_m =_L f_m x_1 \ldots x_n \rightarrow P' x_1 \ldots x_n\}$ is a maximal solution for B in

$$\Gamma, \exists B : K, \exists h : \langle x_1, \ldots, x_j, f_1 x_1 \ldots x_n, \ldots, f_m x_1 \ldots x_n \rangle \in B \rightarrow P' x_1 \ldots x_n .$$

Theorem 8. *Let Γ be a valid Meta context such that Γ has a derivation. Let K be a set type in Γ such that K has the form $\forall x_1 : A_1 \ldots \forall x_n : A_n . Prop$ for some $n > 0$. Let $Q, M_1, \ldots, M_n$ be terms such that the following are derivable.*

$$\begin{array}{l} \Gamma \vdash Q : Prop \\ \Gamma \vdash M_i : [M_1/x_1, \ldots M_{i-1}/x_{i-1}]A_i \quad \text{for } i = 1, \ldots, n \end{array}$$

Then $\{\langle x_1, \ldots, x_n \rangle \mid x_1 =_L M_1 \rightarrow \cdots \rightarrow x_n =_L M_n \rightarrow Q\}$ is a maximal solution for A in $\Gamma, \exists A : K, \exists h : \langle M_1, \ldots, M_n \rangle \in A \rightarrow Q$ and $\{\langle x_1, \ldots, x_n \rangle \mid \neg(x_1 =_L M_1 \wedge \cdots \wedge x_n =_L M_n)\}$ is a maximal solution for A in $\Gamma, \exists A : K, \exists h : \neg(\langle M_1, \ldots, M_n \rangle \in A)$.

Theorem 9. *Let Γ be a valid Meta context such that Γ has a derivation. Let K be a set type in Γ such that K has the form $\forall x_1 : A_1 \ldots \forall x_n : A_n . Prop$ for some $n > 0$. Let Q be a term and A a variable such that A only occurs positively in Q, $\Gamma, \exists A : K \vdash Q : Prop$ is derivable, and there is an h such that $\Gamma, \exists A : K \vdash h : Q$ is derivable. Then $\{\langle x_1, \ldots, x_n \rangle \mid \top\}$ is a maximal solution for A in $\Gamma, \exists A : K, \exists h : Q$.*

Theorem 10. *Let Γ be a valid Meta context such that Γ has a derivation, and let K be a set type in Γ. Let P' and Q' be terms such that $\Gamma, \exists A : K \vdash P' : Prop$ and $\Gamma, \exists A : K \vdash Q' : Prop$ are derivable, P' has the form $\forall y_1 : B_1 \ldots \forall y_q : B_q . P$, and Q' has the form $\forall y_1 : B_1 \ldots \forall y_q : B_q . Q$, where $q \geq 0$ and P and Q are atomic. If A_1 is a maximal solution for A in $\Gamma, \exists A : K, \exists h : P'$ and A_2 is a maximal solution for A in $\Gamma, \exists A : K, \exists h : Q'$, then $A_1 \cap A_2$ is a maximal solution for A in*

$$\Gamma, \exists A : K, \exists h : \forall y_1 : B_1 \ldots \forall y_q : B_q . (P \wedge Q).$$

6 Conclusion

We have shown how to adapt Bledsoe's method for generating maximal solutions for set variables to the Calculus of Constructions. The procedure presented here can be used to help direct an automated search procedure for CC to work efficiently on the class of theorems involving existential quantification over sets. It can also be used as a tactic within an interactive theorem prover to provide some automation for this class of theorems.

We have adapted and generalized results from Bledsoe [2]. The basic rules and combining rules for conjunction were adapted fairly directly, while the combining rules for disjunction were handled in a distributed manner. The remaining rules in Bledsoe [2] are quite specialized and involve substitution instances expressing a function applied n times to x as $f^n(x)$. These rules should also be straightforward to add to the procedure here, though their addition would require adding some axioms to the context to express f^n since it cannot be expressed directly in CC. The procedure is structured in such a way that adding more rules for maximal solutions is achieved by simply adding new search operations.

Future work will first include completing the implementation and installing it as a tactic in a larger system such as Coq. We also plan to adapt other procedures for automating the instantiation of set variables including the $\mathcal{Z}$-match inference rule in [1]. It would also be interesting to extend Bledsoe's procedure directly by transferring our generalization carried out within the type theoretic setting back into the higher-order logic setting. In addition, many other theorem proving techniques have been developed for both higher-order logic and higher-order type theory that would be interesting to investigate and adapt to aid proof search in the other domains.

Acknowledgements The author would like to thank the anonymous referees for valuable comments.

References

1. S. C. Bailin and D. Barker-Plummer. $\mathcal{Z}$-match: An inference rule for incrementally elaborating set instantiation. *Journal of Automated Reasoning*, 11(3):391–428, Dec. 1993.
2. W. W. Bledsoe. A maximal method for set variables in automatic theorem proving. *Machine Intelligence*, 9:53–100, 1979.
3. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
4. R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, 1986.
5. T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, February/March 1988.
6. C. Cornes, J. Courant, J.-C. Filliâtre, G. Huet, P. Manoury, C. Paulin-Mohring, C. Muñoz, C. Murthy, C. Parent, A. Saïbi, and B. Werner. The Coq Proof Assistant reference manual. Technical report, INRIA, 1995.
7. G. Dowek. *Démonstration Automatique dans le Calcul des Constructions*. PhD thesis, L'Université Paris VII, Dec. 1991.

8. G. Dowek. A complete proof synthesis method for the cube of type systems. *Journal of Logic and Computation*, 3(3):287–315, 1993.
9. A. Felty. Encoding the calculus of constructions in a higher-order logic. In *Eighth Annual Symposium on Logic in Computer Science*, pages 233–244, June 1993.
10. M. J. C. Gordon and T. F. Melham. *Introduction to HOL—A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.
11. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, Jan. 1993.
12. W. A. Howard. The formulae-as-type notion of construction, 1969. In *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1980.
13. G. Huet. A uniform approach to type theory. In G. Huet, editor, *Logical Foundations of Functional Programming*. Addison Wesley, 1990.
14. P. Martin-Löf. *Intuitionistic Type Theory*. Studies in Proof Theory Lecture Notes. BIBLIOPOLIS, Napoli, 1984.
15. D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
16. L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Note in Computer Science*. Springer-Verlag, 1994.

A The Complete Search Procedure

In addition to SETVAR and BACKCHAIN in Sect. 4, the following are the remaining search operations of the complete procedure discussed in Sect. 4. In describing these operations, we write $\forall \overline{x}_n : \overline{A}_n . K$ to denote the term $\forall x_1 : A_1 \ldots \forall x_n : A_n . K$. Similarly, we write $\lambda \overline{x}_n : \overline{A}_n . K$ to denote the term $\lambda x_1 : A_1 \ldots \lambda x_n : A_n . K$, and $x\overline{M}_n$ to denote the term $xM_1 \ldots M_n$. In all of these abbreviations, $n \geq 0$.

Definition 11 SPLIT **operation.** Let Γ be a valid Meta context and $\exists z : T$ a candidate variable in Γ. T has the form $\forall \overline{x}_n : \overline{A}_n . x\overline{M}_m$. Let s be *Prop*, *Type*, or *Extern*. If $\Gamma \vdash x\overline{M}_m : s$ and there is a universal variable $w : Q'$ as in BACKCHAIN, then let $Q_1, \ldots, Q_q, Q$ and Γ_1 be as in BACKCHAIN. Choose a j such that $j > 0$. For $i = 1, \ldots, j$, let s_i be either *Prop* or *Type*. For $i = 1, \ldots, j$, let Δ_i be the following context

$$\begin{array}{c} \exists H_i : \forall \overline{x}_n : \overline{A}_n . s_i, \quad \exists K_i : \forall \overline{x}_n : \overline{A}_n . \forall u : H_i \overline{x}_n . s, \\ \forall \overline{x}_n : \overline{A}_n . L = \forall \overline{x}_n : \overline{A}_n . \forall u : H_i \overline{x}_n . K_i \overline{x}_n u, \quad \exists h_{q+i} : \forall \overline{x}_n : \overline{A}_n . H_i \overline{x}_n \end{array}$$

where if $i = 1$, then L is the term $Q(h_1 \overline{x}_n) \ldots (h_q \overline{x}_n)$, and if $i > 1$, then L is the term $K_{i-1} \overline{x}_n (h_{q+i-1} \overline{x}_n)$. Let Γ_2 be the context

$$\forall \overline{x}_n : \overline{A}_n . K_j \overline{x}_n (h_{q+j} \overline{x}_n) = \forall \overline{x}_n : \overline{A}_n . x\overline{M}_m .$$

Apply the substitution $\langle z, (\Gamma_1, \Delta_1, \ldots, \Delta_j, \Gamma_2), \lambda \overline{x}_n : \overline{A}_n . w(h_1 \overline{x}_n) \ldots (h_{q+j} \overline{x}_n) \rangle$ to Γ.

Definition 12 PROD **operation.** Let Γ be a valid Meta context and $\exists z : T$ a candidate variable in Γ. If T has the form $\forall \overline{x}_n : \overline{A}_n . s$ where s is *Type* or *Extern*, then let s' be *Prop* or *Type*, respectively. Apply the substitution $\langle z, \langle \rangle, \lambda \overline{x}_n : \overline{A}_n . s' \rangle$ to Γ.

Definition 13 POLY **operation.** Let Γ be a valid Meta context and $\exists z : T$ a candidate variable in Γ. If T has the form $\forall \overline{x}_n : \overline{A}_n . s$ where s is *Prop*, *Type*, or *Extern*, then let s' be *Prop* or *Type*. Let Δ be the context $\exists h : \forall \overline{x}_n : \overline{A}_n . s', \exists k : \forall \overline{x}_n : \overline{A}_n . \forall u : h\overline{x}_n . s$. Apply the substitution $\langle z, \Delta, \lambda \overline{x}_n : \overline{A}_n . \forall u : h\overline{x}_n . k\overline{x}_n u \rangle$ to Γ.

Note that SPLIT may apply infinitely many ways to the same context, while POLY may create infinite paths.