The Calculus of Constructions as a Framework for Proof Search with Set Variable Instantiation*

Amy Felty
Bell Laboratories
Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974, USA
felty@research.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. 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 show how Bledsoe's techniques can be incorporated into a reformulation of a search procedure for CC given by Dowek and extended to handle terms with dependent types. We introduce a notion of *search context* for CC which allow us to separate the operations of assumption introduction and backchaining. Search contexts allow a smooth integration of the step which finds solutions to set variables. We discuss how the procedure can be restricted to obtain procedures for set variable instantiation in sublanguages of CC such as the Logical Framework (LF) and higher-order hereditary Harrop formulas (hohh). The latter serves as the logical foundation of the λProlog logic programming language.

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 [15] and Isabelle [23] implement higher-order logic, while Coq [8] implements the Calculus of Constructions (CC) type theory [7] and Nuprl [6] implements Martin-Löf type theory [20]. 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 [17]. Although it is less direct for CC, one way to view the correspondence was shown in Felty [12]. Intuitively, a functional type $P \to Q$

*In *Theoretical Computer Science*, 232(1-2):187–229, February 2000.

cation. An important 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 [5]) 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 adapt techniques described in Bledsoe [3] for the automatic discovery of substitutions for set variables to a modified version of the search procedure for CC given by Dowek in [9, 10]. (Below, we refer exclusively to [10] except for the case where we use an auxiliary result that occurs only in [9].) Dowek's procedure actually operates on all type systems in Barendregt's cube [2]. We use only the restriction to CC. In our formulation, 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 [3], 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 [3] 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. The main cause of such infinite branching is the need to enumerate types. There are many ways to direct the search by tuning it to a particular class of theorems. Our work can be viewed as the tuning of Dowek's procedure to find proofs more directly 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.

CC. In Dowek [10], the operations of assumption introduction and backchaining are combined; search contexts allow us to separate them. This separation was inspired by our implementation in λProlog (see below). By making this separation, we are able to present the procedure in more fine-grain steps. We believe this refinement enhances understanding as well as allows a smoother integration of the step which finds maximal solutions to set variables. The integration of this step is the second part of the work. The result is a procedure which incorporates Bledsoe's method into Dowek's algorithm.

We present two procedures. The first, called SetVar, is not complete for CC, but is complete for the class considered by Bledsoe as well as for proof search in interesting sublanguages of CC such as higher-order hereditary Harrop formulas (hohh) [21] and the Logical Framework (LF) [16]. In LF, proof search covers the search for a term of a particular type, but not for a type of a particular kind. We present the SetVar procedure as a set of three search operations, one whose sole purpose is to instantiate set variables. If we leave out this operation, the SetVar procedure restricted to the other two search operations is a complete search procedure for both hohh and LF. However, simply adding in this operation does not present an interesting search procedure for either language. In the case of LF, there are no set variables because quantification over predicates is not allowed, so the extra search operation does not add anything. In the case of hohh, quantification over predicates that correspond to set variables is severely restricted, so the extra search operation adds little. We will discuss how, in both cases, the languages can be directly extended to allow set variables in a manner that is analogous to the way that first-order logic is extended in Bledsoe's system. Furthermore, set variables with dependent types are easily incorporated into LF.

The second procedure, SetVar$^+$, extends SetVar to a complete procedure for CC by adding a few more search operations. As a whole, it can be viewed as a reformulation of Dowek's procedure with the addition of an operation specialized for finding maximal solutions to set variables. The class of variables corresponding to set variables are already contained within CC, and so no extension of the language needs to be made to incorporate them. However, adding the operation which instantiates them provides a procedure which expands branches of the search that lead to maximal solutions more directly. On the other hand, removing this specialized operation does not affect completeness.

This paper extends Felty [14] in several ways. First, we separate the procedures SetVar and SetVar+. SetVar should be more useful in practice because it eliminates the non-determinism that corresponds to enumerating types, while still handling most examples and remaining complete for various sublanguages of CC extended with set variables. Second, the introduction of search contexts is new. Third, we include proofs of soundness of SetVar and soundness and completeness of SetVar$^+$. We prove completeness by showing that every operation in Dowek's procedure has a corresponding set of operations in SetVar$^+$. We could prove soundness by proving the converse, *i.e.*, that every execution of SetVar$^+$ can be divided into sequences of operations such that each sequence corresponds to an operation in Dowek's procedure. Instead, we prove it directly to illustrate how it can be proved using search contexts. The proof follows the basic outline of Dowek's proof and in addition verifies that the additional operation for finding maximal solutions preserves soundness.

We have implemented a prototype of the SetVar procedure in λProlog [21]. We use a goal-directed tactic style framework where each of the search operations of the procedure is implemented as a tactic [13]. The SetVar procedure as described here does not resolve all non-determinism in search. In the prototype, the non-determinism is resolved by having the user ples in this paper as well as some of the examples classified as "major examples" in Bledsoe [3]. Although we have not yet done so, the set of tactics we have implemented can be combined to obtain a procedure that corresponds fairly directly to a one-pass version of Bledsoe's procedure. Such a procedure would be able to prove most of the examples in Bledsoe [3] fully automatically. This procedure 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 due to Dowek [10] which is used as the foundation for the search procedures. We also 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. In addition, we define maximal solutions in our setting, which directly extend those in Bledsoe [3]. In Sect. 3, we present search contexts and use them in presenting the SetVar search procedure. We also show that it is sound. In addition, we prove theorems that justify the maximal solutions used in the search procedure. These theorems are extensions of the theorems in Bledsoe [3]. In Sect. 4, we present the SetVar$^+$ procedure and prove its correctness. Finally, we conclude in Sect. 5.

2 The Calculus of Constructions and Set Variables

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

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

Here *Type* and *Prop* are constants called *sorts*, x ranges over variables, and P and Q range over terms. We also use other upper case letters to denote terms, and both upper and lower case letters to denote variables. 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 sometimes leave off the type P of x when it can be easily inferred. A *context* is an ordered list of pairs of the form $x : P$, called a *declaration*, where x is a variable and P a term. We use Γ, Δ, and Φ to denote contexts.

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 declaration in Γ. A tree built using the rules of Fig. 1 is called a *proof*. We say that Γ is a *valid context* if there is a proof such that ($\vdash \Gamma$ $\mathsf{context}$) occurs at the root. We say that $\Gamma \vdash P : Q$ *holds* or *is derivable* in CC if Γ is a valid context and this judgment occurs at the root of a proof. In this case, we also say that P *has type* Q or *is of type* Q in Γ, that Q *is the type of* P in Γ, and that P is *well-typed* in Γ. When Q is a sort, we say that P is a *type* 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.

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 $=_{\beta\eta}$. Given valid context Γ, all terms that are well-typed in Γ have a unique $\beta\eta$-normal form and a unique $\beta\eta$-long form (which we call the *normal form in* Γ), as well as a unique type modulo

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

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

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

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

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

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

Figure 1: CC Typing Rules

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

$$\frac{\exists x\!:\!P \in \Gamma \quad \vdash \Gamma \ \mathsf{context}}{\Gamma \vdash x : P} \ \text{(Q-INIT)}$$

Figure 2: Additional Typing Rules for CC^+

$\beta\eta$-equivalence. 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.

Several other properties of CC are used later. For example if ($\vdash \Gamma, x\!:\!P$ $\mathsf{context}$) is derivable, we know that $\Gamma \vdash P : s$ is derivable for some sort s. If $\Gamma \vdash \lambda x : R.P : \forall x : R.Q$ is derivable, we know that $\Gamma, x\!:\!R \vdash P : Q$ is derivable. Also if Γ, Δ and Γ, Δ' are valid contexts, then the context Γ, Δ, Δ' is also valid as long as the variables on the left in declarations in Δ and Δ' are distinct. This property is called *thinning*. Finally, we note that for terms P, Q, R, if $P =_{\beta\eta} Q$, then $[R/x]P =_{\beta\eta} [R/x]Q$.

As in Dowek [10], the description of the search procedure and the proof of its correctness relies on extending CC to allows existential quantification of the form $\exists x : P$ and equations between terms, written $P = Q$, to appear in contexts. We call the new inference system CC^+. Given a context Γ, a variable x is *universal (existential) in* Γ if there is a P such that $x\!:\!P \in \Gamma$ ($\exists x : P \in \Gamma$). The declaration $x\!:\!P \in \Gamma$ is also called a *universal declaration* and $\exists x\!:\!P \in \Gamma$ is called an *existential declaration*. The equation $P = Q$ is called a *constraint*. A *context element* is either a declaration or constraint, sometimes denoted e. 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. The CC^+ typing rules include all those for CC plus the additional rules in Fig. 2. In addition, $=_{\beta\eta}$ in (CONV) in Fig. 1 is replaced by $=_{\beta\eta\Gamma}$ which denotes equality modulo $\beta\eta$-conversion plus the constraints in Γ. A *subcontext* of a context Γ is any context obtained by removing some elements of Γ. Given terms P and Q and context Γ, P is said to be *of type Q in Γ without using the constraints* if there is

$$
\begin{array}{rcl}
\vee & := & \lambda A, B\!:\!Prop.\forall C\!:\!Prop.((A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C) \\
\exists_Q & := & \lambda P\!:\!Q \rightarrow Prop.\forall C\!:\!Prop.((\forall x\!:\!Q.Px \rightarrow C) \rightarrow C) \\
\bot & := & \forall C\!:\!Prop.C \\
\top & := & \forall C\!:\!Prop.C \rightarrow C \\
\neg & := & \lambda A\!:\!Prop.A \rightarrow \bot \\
=_Q & := & \lambda M, N\!:\!Q.\forall P\!:\!Q \rightarrow Prop.PM \rightarrow PN
\end{array}
$$

Figure 3: CC Encoding of the Connectives of Higher-Order Logic

a subcontext Δ containing no constraints such that $\Delta \vdash P : Q$. All terms that are well-typed in a context without using the constraints have a unique normal form [10]. The *normal form* of a context is obtained by replacing all types of variables and all members of constraints that are well-typed without using the constraints by their normal forms.

We say that a term P is *atomic in context* Γ (in CC or CC$^+$) if there is a Q such that $\Gamma \vdash P : Q$ is derivable and there is a variable x and 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 K is a type in Γ and K is atomic in Γ.

Generally, proof search in CC$^+$ starts with a context of the form $\Gamma, \exists x\!:\!P$ where Γ is a context of universal declarations, P is a property to be proved from the declarations in Γ, and x is a "placeholder" for a proof of P. The goal of the search process is to instantiate x with a term of type P (or equivalently, a proof of formula P). The search process will generate the instantiation incrementally, and along the way new existential variables and constraints between terms will be generated. Proof search terminates successfully when the term instantiating x contains no existential variables and all constraints generated along the way are satisfied.

It is shown in Huet [18] that higher-order logic is contained within CC. 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 $(\exists_Q \lambda x\!:\!Q.A)$, it represents the term $\forall C\!:\!Prop.((\forall x\!:\!Q.A \rightarrow C) \rightarrow C)$, and encodes the formula $\exists_Q x.A$ where $\exists_Q$ is the existential quantifier at type Q in higher-order logic. In CC, it must be the case that $\Gamma \vdash Q : Prop$ or $\Gamma \vdash Q : Type$ where Γ is the context in which the existentially quantified expression occurs. We often omit the type subscript Q on $\exists_Q$ because it can be inferred from the type of the bound variable in the argument. For readability, we will use infix notation for the binary connectives. Implication and universal quantification are encoded directly using the function arrow and dependent type constructor of CC, respectively. Note that equality is Leibniz equality indexed over types in the same way as existential quantification.

In set theory, from the fact that $a \in \{x : P(x)\}$, it is possible to immediately deduce $P(a)$. In our encoding, we build in this correspondence directly and define sets to be predicates of a certain class of types. Term A is a *set type in context* Γ if $\Gamma \vdash A : Type$ is derivable and A has the form $\forall x_1\!:\!A_1 \ldots \forall x_n\!:\!A_n.Prop$, where $n > 0$ and for $i = 1, \ldots, n$, A_i is a rigid base type or set type in $\Gamma, x_1\!:\!A_1, \ldots, x_{i-1}\!:\!A_{i-1}$. Term M is a *set in context* Γ if $\Gamma \vdash M : A$ and A is a set type in Γ. In our setting, a set variable is actually a term of a certain form. In particular, a set in context Γ of the form $zz_1 \ldots z_n$ where z is an existential variable in Γ, and $z_1, \ldots, z_n$ are distinct universal variables in Γ is called a *set variable in* Γ.

To illustrate, let Γ be the context $Nat\!:Type, 0\!:\!Nat, s\!:\!Nat \rightarrow Nat$. Note that $Nat \rightarrow Prop$,

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

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

Figure 4: CC Encoding of Sets

$(\mathit{Nat}\to\mathit{Prop})\to\mathit{Prop}$, $((\mathit{Nat}\to\mathit{Prop})\to\mathit{Prop})\to\mathit{Prop}$, *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. We use abbreviations for sets and set operations to keep the correspondence with set membership in Bledsoe's work. Fig. 4 contains these abbreviations. We write $=_S$ for set equality.

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

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

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.)

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

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_2AM'', where A and M'' are terms that must be filled in by proving the following two judgments.

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

As in Sect. 1, we take A to be $\{x\mid Px\}$, and so we must prove $\Gamma'\vdash\{x\mid Px\}:\mathit{Nat}\to\mathit{Prop}$ and find a term M'' such that

$$\Gamma'\vdash M'':(\forall x\!:\!\mathit{Nat}.\langle x\rangle\in\{x\mid Px\}\to Px)\wedge(\exists\ \lambda y\!:\!\mathit{Nat}.\langle y\rangle\in\{x\mid Px\})$$

$Nat.Px$ which is η-equivalent to P. After expanding definitions in the second judgment, it is straightforward to fill in M'' and complete the proof.

Fig. 5 shows maximal solutions for variables A and B in various subformulas. A is assumed to occur in context Γ only in the form $\langle M_1, \ldots, M_n\rangle \in A$, and similarly for B. These are the solutions considered by Bledsoe in the form 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. 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 $fx \in B \to P'(x)$ obtaining maximal solution $\{z \mid \forall x(z = fx \to P'(x))\}$. In the CC version, the types of the last r 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. Since our rules are extensions of Bledsoe's rules, we extend the theorems in [3] which justify the role of these rules in determining maximal solutions. The proofs of the extended theorems appear in Sect. 3.3.

3 Proof Search with Set Variable Instantiation

The SetVar procedure is defined using our modified notion of contexts called search contexts. To distinguish them from the notion of context defined in the previous section, we say *standard context* to denote the latter. In Dowek [10] and Felty [14], the search procedure was described as direct operations on standard contexts. We first define the notions of existential triple and constraint triple which replace existential declarations and constraints. An *existential triple* is a tuple of the form (Φ, z, B) where Φ is a standard context containing only universal declarations, z is a variable, and B is a term. A *constraint triple* is a tuple of the form (Φ, P, Q) where Φ is a standard context containing only universal declarations and P and Q are terms. In either case, Φ is called a *local context* and the universal variables in Φ are called *local variables*. A *search context* is an ordered list of universal declarations, existential triples, and constraint triples.

We define an operation *flatten* on context elements of search contexts as follows:

- *flatten*(e) is e if e is a universal declaration.
- *flatten*$((z_1 : A_1, \ldots, z_n : A_n), z, B)$ is $\exists z : (\forall z_1 : A_1 \ldots \forall z_n : A_n.B)$.
- *flatten*$((z_1 : A_1, \ldots, z_n : A_n), P, Q)$ is $(\forall z_1 : A_1 \ldots \forall z_n : A_n.P) = (\forall z_1 : A_1 \ldots \forall z_n : A_n.Q)$.

We extend the *flatten* operation to search contexts in the obvious way: given context Γ, *flatten*(Γ) is the context such that each element e of Γ is mapped to *flatten*(e). We write $\overline{e}$ as shorthand for *flatten*(e) and $\overline{\Gamma}$ as shorthand for *flatten*(Γ). Note that *flatten* maps a search context to a standard context. We say that a search context Γ is *valid* if $\overline{\Gamma}$ is valid. Note that variables can be renamed so that we can assume that all universal variables and local variables occur at most once on the left of a declaration. We do not do so, but instead assume that all local variables in a particular existential or constraint triple, although not necessarily distinct from local variables in other triples, are distinct from each other and from all other universal variables in the context. Note that under this assumption, given a valid search context $\Gamma, (\Phi, z, B)$ or $\Gamma, (\Phi, P, Q)$, the search context Γ, Φ is also valid and equivalently the standard context $\overline{\Gamma}, \Phi$ is valid.

Subformula		Solution for A or B
1. $\langle x_1, \ldots, x_p \rangle \in A z_1 \ldots z_n \rightarrow P x_1 \ldots x_p$	$\longrightarrow$	$\{\langle x_1, \ldots, x_p \rangle \mid P x_1 \ldots x_p\}$
2. $\langle x_1, \ldots, x_j, f_1 x_1 \ldots x_p, \ldots, f_r x_1 \ldots x_p \rangle \in B z_1 \ldots z_n \rightarrow P' x_1 \ldots x_p$		
	$\longrightarrow$	$\{\langle x_1, \ldots, x_j, w_1, \ldots, w_r \rangle \mid$ $\forall x_{j+1}\!:\!D_{j+1} \ldots \forall x_p\!:\!D_p . w_1 =_{C_{j+1}} f_1 x_1 \ldots x_p$ $\rightarrow \cdots \rightarrow w_r =_{C_{j+r}} f_r x_1 \ldots x_p \rightarrow P' x_1 \ldots x_p\}$
3. $\langle x_1, \ldots, x_j, M_1, \ldots, M_r \rangle \in B z_1 \ldots z_n \rightarrow Q$	$\longrightarrow$	$\{\langle x_1, \ldots, x_j, w_1, \ldots, w_r \rangle \mid$ $w_1 =_{C_{j+1}} M_1 \rightarrow \cdots \rightarrow w_r =_{C_{j+r}} M_r \rightarrow Q\}$
4. $\neg(\langle x_1, \ldots, x_j, M_1, \ldots, M_r \rangle \in B z_1 \ldots z_n)$	$\longrightarrow$	$\{\langle x_1, \ldots, x_j, w_1, \ldots, w_r \rangle \mid$ $\neg(w_1 =_{C_{j+1}} M_1 \wedge \cdots \wedge w_r =_{C_{j+r}} M_r)\}$
5. $\langle N_1, \ldots, N_p \rangle \in A z_1 \ldots z_n$	$\longrightarrow$	$\{\langle x_1, \ldots, x_p \rangle \mid \top\}$
6. If 1-4 yield $\{\langle y_1, \ldots, y_q \rangle \mid Q'\}$, and w is a free variable of type C in Q'		
	$\longrightarrow$	$\{\langle y_1, \ldots, y_q \rangle \mid (\exists\ \lambda w\!:\!C . Q')\}$

Provisos:

- $A z_1 \ldots z_n$ and $B z_1 \ldots z_n$ are set variables in some context Γ, *i.e.*, they are sets in Γ, A and B are existential variables in Γ, and $z_1, \ldots, z_n$ are distinct universal variables in Γ.
- $p > 0, j \geq 0, p > j, r > 0, n \geq 0$.
- $\Gamma \vdash A z_1 \ldots z_n : \forall x_1\!:\!C_1 \ldots \forall x_p\!:\!C_p . \mathit{Prop}$
- $\Gamma \vdash B z_1 \ldots z_n : \forall x_1\!:\!C_1 \ldots \forall x_j\!:\!C_j . C_{j+1} \rightarrow \cdots \rightarrow C_{j+r} \rightarrow \mathit{Prop}$
- $\Gamma \vdash P : \forall x_1\!:\!C_1 \ldots \forall x_p\!:\!C_p . \mathit{Prop}$
- $\Gamma \vdash P' : \forall x_1\!:\!C_1 \ldots \forall x_j\!:\!C_j . \forall x_{j+1}\!:\!D_{j+1} \ldots \forall x_p\!:\!D_p . \mathit{Prop}$
- $\Gamma \vdash f_i : \forall x_1\!:\!C_1 \ldots \forall x_j\!:\!C_j . \forall x_{j+1}\!:\!D_{j+1} \ldots \forall x_p\!:\!D_p . C_{j+i}$ for $i = 1, \ldots, r$
- $\Gamma \vdash Q : \mathit{Prop}$
- $\Gamma, x_1\!:\!C_1, \ldots, x_j\!:\!C_j \vdash M_i : C_{j+i}$ for $i = 1, \ldots, r$
- $\Gamma \vdash N_i : [N_1/x_1, \ldots, N_{i-1}/x_{i-1}]C_i$ for $i = 1, \ldots, p$
- $\Gamma \vdash C : \mathit{Prop}$ or $\Gamma \vdash C : \mathit{Type}$
- All universal variables occurring in $P, P', Q, f_1, \ldots, f_r, M_1, \ldots, M_r$ appear before A or B in Γ.
- $A, B, x_1, \ldots, x_p, w_1, \ldots, w_r$ do not occur free in $P, P', Q, f_1, \ldots, f_r, M_1, \ldots, M_r$.
- A, B also do not occur free in $C_1, \ldots, C_p$ or $C_1, \ldots, C_{j+r}, D_{j+1}, \ldots, D_p$.
- $x_1, \ldots, x_p, w$ are distinct universal variables in Γ that do not occur free elsewhere in Γ.

Figure 5: Maximal Solutions for Various Subformulas

empty by viewing $(\langle\rangle, z, B)$ as alternate syntax for $\exists z : B$ and $(\langle\rangle, P, Q)$ as alternate syntax for $P = Q$. Thus, all standard contexts can be viewed as search contexts of a particular form. This equivalence allows us to directly adapt many properties of contexts shown in [10].

The definition of normal form for a context (see Sect. 2) is extended to search contexts: the *normal form* of a search context Γ is obtained as follows.

- For each universal declaration in Γ, if the type of the universal variable is well-typed in $\overline{\Gamma}$ without using the constraints, replace the type by its normal form in $\overline{\Gamma}$.
- For each existential triple (Φ, z, B) in Γ, if the type of z in *flatten*(Φ, z, B) is well-typed in $\overline{\Gamma}$ without using the constraints, then replace B and the types of the universal variables in Φ with their normal forms in $\overline{\Gamma}, \Phi$.
- For each constraint triple (Φ, P, Q) in Γ, if the members of the constraint *flatten*(Φ, P, Q) are well-typed in $\overline{\Gamma}$ without using the constraints, then replace P, Q, and the types of the universal variables in Φ with their normal forms in $\overline{\Gamma}, \Phi$.

We define substitution for search contexts. Let σ be a set of tuples of the form $\langle z, \Delta, M\rangle$ where z is a variable, Δ is a search context, and M is a term. The set σ is a *substitution* if for any variable z, there is at most one tuple in σ with z as its first component. The application of such a substitution to a term is defined in the usual way ignoring the middle arguments of tuples. The application of substitution σ to a search context Γ, denoted $\sigma\Gamma$, is defined recursively as follows.

- If Γ is $\langle\rangle$, $\sigma\Gamma$ is $\langle\rangle$.
- If Γ is $\Gamma', x : P$, then $\sigma\Gamma$ is $\sigma\Gamma', x : \sigma P$.
- If Γ is $\Gamma', ((z_1 : A_1, \ldots, z_n : A_n), z, B)$ where $n \geq 0$, then if there is a tuple $\langle z, \Delta, M\rangle$ in σ, $\sigma\Gamma$ is $\sigma\Gamma', \Delta$. Otherwise, $\sigma\Gamma$ is $\sigma\Gamma', ((z_1 : \sigma A_1, \ldots, z_n : \sigma A_n), z, \sigma B)$.
- If Γ is $\Gamma', ((z_1 : A_1, \ldots, z_n : A_n), P, Q)$, then $\sigma\Gamma$ is $\sigma\Gamma', ((z_1 : \sigma A_1, \ldots, z_n : \sigma A_n), \sigma P, \sigma Q)$.

By restricting the above definition so that both Γ and Δ are required to be standard contexts, we obtain the definition of substitution given in Dowek [10]. Given substitution σ, we write $\overline{\sigma}$ to denote the substitution obtained by replacing the context argument Δ of each tuple in σ by $\overline{\Delta}$. Note that σ and $\overline{\sigma}$ are the same substitution on terms, *i.e.*, for any term P, $\sigma P = \overline{\sigma} P$.

A valid context Γ is a *success context* if it contains no existential triples and for every constraint triple e, *flatten*(e) relates $\beta\eta$-convertible terms. A valid context Γ is a *failure context* if it contains a constraint triple e such that *flatten*(e) relates two terms that have no free occurrences of existential variables and that are not $\beta\eta$-convertible. Let Γ be a valid search context. A *candidate triple* of Γ is an existential triple

$$((z_1 : A_1, \ldots, z_n : A_n), z, \forall x_1 : B_1 \ldots \forall x_m : B_m . x M_1 \ldots M_p)$$

where $n, m, p \geq 0$ and x is universal in $\Gamma, z_1 : A_1, \ldots, z_n : A_n, x_1 : B_1, \ldots, x_m : B_m$. As we will see in Sect. 3.2, if a valid context is not a success or failure context, there is always at least one candidate triple.

The SETVAR, INTRO, and BACKCHAIN operations described below define the SetVar search procedure. At each step, an operation is applied to a search context in normal form. The result is a substitution σ. The substitution is applied to the input search context which is then normalized to obtain the input to the next step of the procedure. Generally, the original input has the form $\Gamma, (\langle\rangle, z, P)$ where Γ is a standard context and P is a theorem for which a proof is sought. If a success context is reached then the series of substitutions provides a solution to z which represents the proof. Along the way set variables may arise. Their solutions can also be extracted from the series of substitutions. In describing these operations, we often write $\forall \overline{x}_n : \overline{A}_n.K$ to denote the term $\forall x_1 : A_1 \ldots \forall x_n : A_n.K$, where $n \geq 0$. Similarly, we write $\lambda \overline{x}_n : \overline{A}_n.K$ to denote the term $\lambda x_1 : A_1 \ldots \lambda x_n : A_n.K$. Note that this notation is overloaded since it also denotes *flatten*. However, since *flatten* only applies to contexts or context elements, there should be no confusion.

SETVAR operation. Let Γ be a valid search context and $((z_1 : A_1, \ldots, z_n : A_n), z, \forall x_1 : C_1 \ldots \forall x_p : C_p.Prop)$ a candidate triple in Γ, where $n \geq 0$, $p > 0$, and $\forall x_1 : C_1 \ldots \forall x_p : C_p.Prop$ is a set type. Let Φ be the context $z_1 : A_1, \ldots, z_n : A_n$. In order for this operation to apply, there must be q occurrences of z in terms in Γ where $q > 0$, and for $i = 1, \ldots, q$, the i^{th} occurrence is in some term P_i which is part of an existential triple of the form $((\Phi, \Phi_i), z'_i, P_i)$ occurring after the candidate triple containing z. Furthermore, P_i must be of one of the following forms:

1. $\langle x_1, \ldots, x_p \rangle \in z z_1 \ldots z_n \to P x_1 \ldots x_p$
2. $\langle x_1, \ldots, x_j, f_1 x_1 \ldots x_{p'}, \ldots, f_r x_1 \ldots x_{p'} \rangle \in z z_1 \ldots z_n \to P' x_1 \ldots x_{p'}$
3. $\langle x_1, \ldots, x_j, M_1, \ldots, M_r \rangle \in z z_1 \ldots z_n \to Q$
4. $\neg(\langle x_1, \ldots, x_j, M_1, \ldots, M_r \rangle \in z z_1 \ldots z_n)$
5. $\langle N_1, \ldots, N_p \rangle \in z z_1 \ldots z_n$

such that the provisos of the corresponding rule in Fig. 5 hold in the context $\overline{\Gamma}, z_1 : A_1, \ldots, z_n : A_n$. For $i = 1, \ldots, q$, let Q_i be the solution for $z z_1 \ldots z_n$ 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 $\langle z, \langle\rangle, \lambda \overline{z}_n : \overline{A}_n.Q \rangle$.

INTRO operation. Let Γ be a valid search context and $((z_1 : A_1, \ldots, z_n : A_n), z, \forall x : A.B)$ a candidate triple in Γ. Let z' be a variable that does not occur in Γ and assume x does not occur in Γ. Let Δ be the context containing the single triple $((z_1 : A_1, \ldots, z_n : A_n, x : A), z', B)$, and let σ be $\{\langle z, \Delta, z' \rangle\}$.

BACKCHAIN operation. Let Γ be a valid search context and $((z_1 : A_1, \ldots, z_n : A_n), z, x M_1 \ldots M_m)$ a candidate triple in Γ, where $m, n \geq 0$, and $\overline{\Gamma}, z_1 : A_1, \ldots, z_n : A_n \vdash x M_1 \ldots M_m : s$ holds where s is *Prop* or *Type*. If there is a universal declaration $w : Q$ such that either w is one of $z_1, \ldots, z_n$ or $w : Q$ occurs to the left of $((z_1 : A_1, \ldots, z_n : A_n), z, x M_1 \ldots M_m)$ in Γ, the judgment $\overline{\Gamma}, z_1 : A_1, \ldots, z_n : A_n \vdash Q : s$ holds, Q has the form $\forall y_1 : Q_1 \ldots \forall y_q : Q_q.y N_1 \ldots N_p$ $(p, q \geq 0)$, and y is x or any existential variable in Γ, then we can "backchain" on Q as follows. Let $h_1, \ldots, h_q$ be variables that do not occur in Γ. Let Φ be the context $z_1 : A_1, \ldots, z_n : A_n$. Let Δ be the context

$$
\begin{array}{l}
(\Phi, h_1, Q_1), \\
(\Phi, h_2, [h_1 z_1 \ldots z_n / y_1] Q_2), \\
\quad \vdots \\
(\Phi, h_q, [h_1 z_1 \ldots z_n / y_1, \ldots, h_{q-1} z_1 \ldots z_n / y_{q-1}] Q_q), \\
(\Phi, [h_1 z_1 \ldots z_n / y_1, \ldots, h_q z_1 \ldots z_n / y_q] y N_1 \ldots N_n, x M_1 \ldots M_m).
\end{array}
$$

A *derivation* of a search context Γ is a list of substitutions $\sigma_1, \ldots, \sigma_n$ such that for $i = 1, \ldots, n$, σ_i is the result of applying one of the search operations to the normal form of $\sigma_{i-1} \ldots \sigma_1 \Gamma$ and the normal form of $\sigma_n \ldots \sigma_1 \Gamma$ is a success context.

As mentioned earlier, the use of search contexts allows us to separate a single operation in Dowek's procedure into two operations here, INTRO and BACKCHAIN, which correspond to fairly intuitive steps of proof search. The INTRO operation performs the introduction of assumptions into the environment. In particular, assumptions are introduced into local contexts. In the search context as a whole, the third element of existential triples represent the formulas that must be proved, and for any given formula the assumptions that are available to use in its proof are those in its local context as well as all universal declarations that occur before the triple.

The BACKCHAIN operation performs the usual operation of backchaining on an assumption when the formula to be proved "matches" the atomic part of the assumption. In particular, for the particular candidate triple involved, it is not required that its third argument be the same as or unify with the atomic part of the assumption used in backchaining. Instead, a constraint is added that must be checked as the search proceeds. Subsequent search operations may instantiate existential variables in such a way that the constraint may or may not relate two terms that are $\beta\eta$-convertible. In the case when they are not $\beta\eta$-convertible, the context becomes a failure context. In addition to the constraint, BACKCHAIN generates new existential triples for the subgoals that must still be proven. Also, a substitution is formed which instantiates the existential variable in the original candidate triple. Whenever there are subgoals, this instantiation is partial since it will contain occurrences of the new existential variables created for the subgoals.

Search contexts provide a way to simplify the handling of scoping constraints within the framework of a proof search procedure which operates by filling in substitution instances for existential variables incrementally. This notion of context does not deviate far from the standard one in the sense that at any point during search a simple translation via the *flatten* operation can be applied to transform search contexts which contain local contexts back to ordinary contexts. This *flatten* operation is essential in forming and propagating substitutions. These substitutions can be said to use a *functional* encoding of scope. In Dowek's procedure [10], this functional encoding of scope is used in both contexts and substitutions. At the other end of the spectrum are various calculi that avoid a functional encoding by integrating existential variables with explicit substitutions. Examples include the $\lambda\sigma$-calculus [11], the $\lambda_{\mathcal{C}\Pi}$-calculus [22], and the substitution calculus for Martin-Löf type theory [19]. In these calculi, existential variables are distinct from ordinary variables and substitutions are represented explicitly, allowing reduction of terms with existential variables to be delayed as necessary until the terms are filled in. The calculi involved are more complex, but they provide simplified handling of scoping constraints and representation of substitutions.

Note that the procedure as described is non-deterministic since it does not specify an order on the application of search operations. As mentioned earlier, our implementation in λProlog resolves non-determinism by requesting input from the user. Depth-first search with backtracking is another possible strategy.

To illustrate, we describe the execution of the procedure on two examples. We start with a simple example to illustrate the interaction of INTRO and BACKCHAIN. The second example is a modified form of our earlier example. The proof of this example contains an essential use of the SETVAR operation; it is not possible to prove it using only INTRO and BACKCHAIN. For the

$$Nat\colon Type, P\colon Nat \to Prop, a\colon Nat$$

as in the previous section from which we want to prove the theorem $(\forall n : Nat.Pn) \to Pa$. We begin with the following search context.

$$\Gamma, (\langle\rangle, M, (\forall n\colon Nat.Pn) \to Pa) \tag{1}$$

This context is in normal form and the existential triple is a candidate triple to which the INTRO operation can be applied. Note that $(\forall n\colon Nat.Pn) \to Pa$ can be written $\forall h\colon (\forall n\colon Nat.Pn).Pa$. The operation results in a substitution σ_1 of the form

$$\{\langle M, ((h\colon (\forall n\colon Nat.Pn)), M', Pa), M'\rangle\}$$

where M' is a new variable. Applying this substitution to (1), we obtain the context

$$\Gamma, ((h\colon (\forall n\colon Nat.Pn)), M', Pa). \tag{2}$$

When applying INTRO, it is actually not necessary to change the name of the existential variable. Here, all occurrences of M are replaced with M' which is another variable of the same type (after applying *flatten*). Instead, we can just keep M. We adopt this convention in the next example below. In this example, we can now apply BACKCHAIN with the existential triple as the candidate triple. The universal declaration we will use in this application of BACKCHAIN is $h\colon (\forall n\colon Nat.Pn)$. We know this operation can be applied because both of the following judgments hold as required.

$$\Gamma, \exists M'\colon (\forall n\colon Nat.Pn) \to Pa, h\colon (\forall n\colon Nat.Pn) \vdash Pa : Prop$$
$$\Gamma, \exists M'\colon (\forall n\colon Nat.Pn) \to Pa, h\colon (\forall n\colon Nat.Pn) \vdash (\forall n\colon Nat.Pn) : Prop$$

We form the context Δ_2 of the BACKCHAIN operation

$$((h\colon (\forall n\colon Nat.Pn)), N, Nat), ((h\colon (\forall n\colon Nat.Pn)), [Nh/n]Pn, Pa)$$

where the first element is an existential triple with new variable N, and the second element is a constraint triple. The term $[Nh/n]Pn$ is just $P(Nh)$. The substitution σ_2 of this operation is

$$\{\langle M', \Delta_2, \lambda h\colon (\forall n\colon Nat.Pn).h(Nh)\rangle\}.$$

Applying σ_2 to (2) completes the application of BACKCHAIN and gives

$$\Gamma, ((h\colon (\forall n\colon Nat.Pn)), N, Nat), ((h\colon (\forall n\colon Nat.Pn)), P(Nh), Pa). \tag{3}$$

Let Γ' denote the above context. Note that $\overline{\Gamma'}$ is

$$\Gamma, \exists N\colon \forall h\colon (\forall n\colon Nat.Pn).Nat, \forall h\colon (\forall n\colon Nat.Pn).P(Nh) = \forall h\colon (\forall n\colon Nat.Pn).Pa.$$

One more application of BACKCHAIN will complete the search. This time, we apply it using the candidate triple $((h : (\forall n : Nat.Pn)), N, Nat)$ and the universal declaration $a\colon Nat$. In this case, the two typing judgments required to hold in order to apply BACKCHAIN are the same:

$$\overline{\Gamma'}, h\colon (\forall n\colon Nat.Pn) \vdash Nat : Type.$$

The context Δ_3 contains only the simple constraint $((h : (\forall n : Nat.Pn)), Nat, Nat)$ and thus, the substitution σ_3 is $\{\langle N, \Delta_3, \lambda h : (\forall n : Nat.Pn).a\rangle\}$. Applying this substitution to (3) and normalizing results in the following context.

$$\Gamma, ((h\colon (\forall n\colon Nat.Pn)), Nat, Nat), ((h\colon (\forall n\colon Nat.Pn)), Pa, Pa). \tag{4}$$

equivalent terms. Thus it is a success context and search is completed. The proof of the formula $(\forall n : Nat.Pn) \rightarrow Pa$ in the context we started with is obtained by applying the substitutions obtained at each step to the original existential variable M and normalizing. In this case, the normal form of $\sigma_3\sigma_2\sigma_1 M$ is the term $\lambda h : (\forall n : Nat.Pn).ha$

For the second example, let Γ be the context

$$Nat : Type, P : Nat \rightarrow Prop, Q : Nat \rightarrow Prop.$$

We want to find a term to instantiate M in the following search context.

$$\Gamma, (\langle\rangle, M, \exists\ \lambda A : Nat \rightarrow Prop.((\forall x : Nat.\langle x\rangle \in A \rightarrow Px) \wedge (\forall x : Nat.\langle x\rangle \in A \rightarrow Qx))) \tag{5}$$

The formula we want to prove contains occurrences of $\exists$ and $\wedge$, which we must first expand before proceeding with search. To simplify the presentation of this example, we first make some observations about proofs of formulas containing these connectives. Consider the general case of proof search in a context of the form $\Gamma, (\Phi, M, \exists\ \lambda x : Q.P)$ where Φ is a context of the form $z_1 : A_1, \ldots, z_n : A_n$. Expanding $\exists$, this context is the same as

$$\Gamma, (\Phi, M, \forall C : Prop.(\forall x : Q.Px \rightarrow C) \rightarrow C). \tag{6}$$

In general, in searching for a proof of an existential formula, a term is chosen to instantiate the bound variable and search proceeds. Alternately, a variable or placeholder is used which gets filled in as search continues. In the SetVar procedure, two applications of INTRO followed by an application of BACKCHAIN to the existential triple in context (6) has the affect of introducing such a placeholder. To see this, first note, that we can apply INTRO, generating the substitution σ_1,

$$\{\langle M, ((\Phi, C : Prop), M, (\forall x : Q.Px \rightarrow C) \rightarrow C), M\rangle\}.$$

Here, we reuse the name M as discussed above. Applying σ_1 to (6) results in the context

$$\Gamma, ((\Phi, C : Prop), M, (\forall x : Q.Px \rightarrow C) \rightarrow C). \tag{7}$$

A second INTRO generates the substitution σ_2,

$$\{\langle M, ((\Phi, C : Prop, h : (\forall x : Q.Px \rightarrow C)), M, C), M\rangle\}$$

and thus the context

$$\Gamma, ((\Phi, C : Prop, h : (\forall x : Q.Px \rightarrow C)), M, C) \tag{8}$$

Now, we can apply BACKCHAIN to the above existential triple using universal declaration $h : (\forall x : Q.Px \rightarrow C)$. From now on, we leave out showing that the necessary typing judgments hold in order for BACKCHAIN to be applicable. Using new variables X and M', we form the context Δ_3 as follows

$$\begin{array}{l} ((\Phi, C : Prop, h : (\forall x : Q.Px \rightarrow C)), X, Q), \\ ((\Phi, C : Prop, h : (\forall x : Q.Px \rightarrow C)), M', P(Xz_1 \ldots z_n Ch)), \\ ((\Phi, C : Prop, h : (\forall n : Nat.Pn)), C, C). \end{array}$$

The substitution σ_3 of this operation is

$$\{\langle M, \Delta_3, \forall \overline{z}_n : \overline{A}_n.\lambda C : Prop.\lambda h : (\forall x : Q.Px \rightarrow C).h(Xz_1 \ldots z_n Ch)(M'z_1 \ldots z_n Ch)\rangle\}.$$

Applying σ_3 to (8), we get Γ, Δ_3. Note the roles of X and M'. In particular, $Xz_1 \ldots z_n Ch$ is the placeholder for the term bound by existential quantification while $M'z_1 \ldots z_n Ch$ must be three steps. They are introduced only to be used immediately in backchaining and it is unlikely that they will have any further role in the search for a proof. Also, note that the constraint in Δ_3 relates equivalent terms and that no subsequent instantiations of existential variables will change that. We use these facts to introduce a search operation, which we call EXISTS-INTRO, that abbreviates this sequence of steps and eliminates C, h, and the constraint. In particular, we introduce a new constant $\exists I$. From a context of the form in (6) using new variables X_0 and M_0', the EXISTS-INTRO operation generates the context Δ

$$(\Phi, X_0, Q), (\Phi, M_0', P(X_0 z_1 \ldots z_n))$$

and the substitution σ

$$\{\langle M, \Delta, \forall \overline{z}_n : \overline{A}_n . (\exists I\ (X_0 z_1 \ldots z_n)\ (M_0' z_1 \ldots z_n))\rangle\}.$$

In our example, we will use this operation in place of the sequence of two applications of INTRO followed by an application of BACKCHAIN as above. It will always be the case that any application of this operation can be expanded into a sequence of the three operations using new variables X, M', C, and h. In the abbreviated version all occurrences of $X_0 z_1 \ldots z_n$ and $M_0' z_1 \ldots z_n$ stand for $X z_1 \ldots z_n C h$ and $M' z_1 \ldots z_n C h$, respectively. Also, $\exists I\ (X_0 z_1 \ldots z_n)\ (M_0' z_1 \ldots z_n)$ abbreviates the term

$$\lambda C : Prop . \lambda h : (\forall x : Q . Px \to C) . h (X z_1 \ldots z_n C h)(M' z_1 \ldots z_n C h).$$

Since the variables C and h along with their types are left out of local contexts, these declarations as well as the constraint must be put back in to get the expanded form. In the unabbreviated sequence, note that once C and h are introduced, they stay around. Thus, the abbreviated form actually changes the contexts that appear in subsequent search. However, it is straightforward to transform a derivation that uses EXISTS-INTRO to one containing only SETVAR, INTRO, and BACKCHAIN, systematically adding occurrences of C and h where necessary. Using the abbreviated form has the consequence of imposing the restriction that, because C and h do not appear at all, they do not apper in subsequent substitution terms. This restriction is not a serious one for the class of theorems we are considering.

We introduce a similar operation called AND-INTRO to abbreviate several steps for the case when the context has the form $\Gamma, (\Phi, M, A \wedge B)$. Note that this context denotes

$$\Gamma, (\Phi, M, \forall C : Prop . (A \to B \to C) \to C).$$

AND-INTRO generates the context Δ, which is simply

$$(\Phi, M_0, A), (\Phi, M_0', B)$$

and the substitution σ

$$\{\langle M, \Delta, \forall \overline{z}_n : \overline{A}_n . (\wedge I\ (M_0 z_1 \ldots z_n)(M_0' z_1 \ldots z_n))\rangle\}.$$

This operation can also be expanded to two applications of INTRO followed by BACKCHAIN. Similar to EXISTS-INTRO, there are variables M, M', C, and h such that in the abbreviated version, all occurrences of $M_0 z_1 \ldots z_n$ and $M_0' z_1 \ldots z_n$ stand for $M z_1 \ldots z_n C h$ and $M' z_1 \ldots z_n C h$, respectively, and $\wedge I\ (M_0 z_1 \ldots z_n)(M_0' z_1 \ldots z_n)$ abbreviates the term

$$\lambda C : Prop . \lambda h : A \to B \to C . h (M z_1 \ldots z_n C h)(M' z_1 \ldots z_n C h).$$

added to local context Φ in elements of Δ and the constraint $((\Phi, C\!:\!Prop, h\!:\!A \to B \to C), C, C)$ must also be added to Δ. Also, AND-INTRO imposes a restriction similar to EXISTS-INTRO since C and h do not apper in Φ.

The EXISTS-INTRO and AND-INTRO operations, respectively, can now be used for the first two steps of proof search in our second example denoted by the context (5). First, the result of applying EXISTS-INTRO is the context Δ_1 and substitution σ_1, respectively, as follows where A_0 and M_0' are new variables.

$$\begin{array}{lcl}\Delta_1 & := & (\langle\rangle, A_0, \mathit{Nat} \to \mathit{Prop}), (\langle\rangle, M_0', (\forall x\!:\!\mathit{Nat}.\langle x\rangle \in A_0 \to Px) \wedge (\forall x\!:\!\mathit{Nat}.\langle x\rangle \in A_0 \to Qx)) \\ \sigma_1 & := & \{\langle M, \Delta_1, (\exists I\ X_0\ M_0')\rangle\}\end{array}$$

Applying σ_1 to context (5), we get the following context:

$$\Gamma, (\langle\rangle, A_0, \mathit{Nat} \to \mathit{Prop}), (\langle\rangle, M_0', (\forall x\!:\!\mathit{Nat}.\langle x\rangle \in A_0 \to Px) \wedge (\forall x\!:\!\mathit{Nat}.\langle x\rangle \in A_0 \to Qx)). \tag{9}$$

Using the triple containing M_0' as the candidate triple, the result of applying AND-INTRO is the following context and substitution:

$$\begin{array}{lcl}\Delta_2 & := & (\langle\rangle, M_1', \forall x\!:\!\mathit{Nat}.\langle x\rangle \in A_0 \to Px), (\langle\rangle, M_2', \forall x\!:\!\mathit{Nat}.\langle x\rangle \in A_0 \to Qx) \\ \sigma_2 & := & \{\langle M_0', \Delta_2, (\wedge I\ M_1'\ M_2')\rangle\}.\end{array}$$

Applying σ_2 to context (9), we get the following context:

$$\begin{array}{l}\Gamma, (\langle\rangle, A_0, \mathit{Nat} \to \mathit{Prop}), \\ (\langle\rangle, M_1', \forall x\!:\!\mathit{Nat}.\langle x\rangle \in A_0 \to Px), (\langle\rangle, M_2', \forall x\!:\!\mathit{Nat}.\langle x\rangle \in A_0 \to Qx).\end{array} \tag{10}$$

We can now apply SETVAR to obtain a solution for A_0 using the maximal solutions for the two types containing A_0. In particular, for this application, the existential triple containing A_0 is the candidate triple and the remaining two existential triples contain occurrences of A_0. Both occurrences are in formulas of the first form listed in the definition of SETVAR and thus the maximal solution in each case is obtained using rule 1 of Fig. 5. The substitution resulting from this application is:

$$\sigma_3 := \{\langle A_0, \langle\rangle, \{\langle x\rangle \mid Px\} \cap \{\langle x\rangle \mid Qx\}\rangle\}.$$

After substitution and β-conversion, the context becomes:

$$\begin{array}{l}\Gamma,(\langle\rangle, M_1', \forall x\!:\!\mathit{Nat}.(\langle x\rangle \in (\{\langle x\rangle \mid Px\} \cap \{\langle x\rangle \mid Qx\})) \to Px), \\ \quad (\langle\rangle, M_2', \forall x\!:\!\mathit{Nat}.(\langle x\rangle \in (\{\langle x\rangle \mid Px\} \cap \{\langle x\rangle \mid Qx\})) \to Qx).\end{array}$$

Note that expanding all definitions, this context is equivalent to

$$\begin{array}{l}\Gamma,(\langle\rangle, M_1', \forall x\!:\!\mathit{Nat}.(\forall C\!:\!\mathit{Prop}.((Px \to Qx \to C) \to C)) \to Px), \\ \quad (\langle\rangle, M_2', \forall x\!:\!\mathit{Nat}.(\forall C\!:\!\mathit{Prop}.((Px \to Qx \to C) \to C)) \to Qx).\end{array}$$

From this point on, several more instances of INTRO and BACKCHAIN are needed to transform this context to a success context.

To see why this derivation can not be completed without SETVAR, consider again the context (10) of this example just before SETVAR was applied. Expanding definitions, this context is equivalent to

$$\Gamma, (\langle\rangle, A_0, \mathit{Nat} \to \mathit{Prop}), (\langle\rangle, M_1', \forall x\!:\!\mathit{Nat}.A_0 x \to Px), (\langle\rangle, M_2', \forall x\!:\!\mathit{Nat}.A_0 x \to Qx) \tag{11}$$

$$\Gamma, ((x\!:\!Nat), A_0, Prop), ((x\!:\!Nat, h\!:\!A_0x), M_1', Px), ((x\!:\!Nat, h\!:\!A_0x), M_2', Qx) \quad (12)$$

At this point BACKCHAIN can be applied to any of the existential triples, but none leads to a success context. For example, consider the first triple. The only universal declarations that can be used in backchaining are the declarations of $P\!:\!Nat \to Prop$ or $Q\!:\!Nat \to Prop$. If the first is used, then the following context and substitution are generated.

$$\Delta := ((x\!:\!Nat), X, Nat), ((x\!:\!Nat), Prop, Prop)$$
$$\sigma := \{\langle A_0, \Delta, \lambda x\!:\!Nat.P(Xx)\rangle\}$$

After one more BACKCHAIN to fill in X using local declaration $x\!:\!Nat$, the instantiation for A_0 becomes $\lambda x : Nat.Px$. Similarly, if the declaration $Q : Nat \to Prop$ were chosen instead, two applications of BACKCHAIN would lead to the instance $\lambda x\!:\!Nat.Qx$ for A_0.

The same problem occurs if we begin with a BACKCHAIN using the second or third existential triples in context (12). Consider the second triple. The only universal declaration that can be used in backchaining is $h\!:\!A_0x$ in the local context. Using this declaration, the following context and substitution are generated.

$$\Delta := ((x\!:\!Nat, h\!:\!A_0x), A_0x, Px)$$
$$\sigma := \{\langle M_1', \Delta, \lambda x\!:\!Nat.\lambda h\!:\!A_0x.h\rangle\}$$

Applying σ to the context (12) results in the context

$$\Gamma, ((x\!:\!Nat), A_0, Prop), ((x\!:\!Nat, h\!:\!A_0x), A_0x, Px), ((x\!:\!Nat, h\!:\!A_0x), M_2', Qx)$$

The variable A_0 will not get filled in until the first existential triple is used in backchaining. The only way to satisfy the new constraint is to use $P\!:\!Nat \to Prop$ as the universal declaration in such an application of BACKCHAIN, which as before, leads to $\lambda x\!:\!Nat.Px$ as the instantiation for A_0. At this point, the only way to continue search is to apply BACKCHAIN to the existential triple containing M_2'. However, such a BACKCHAIN leads to a constraint $((x : Nat, h : A_0x), Px, Qx)$ which relates two terms that are not $\beta\eta$-convertible, and thus the result is a failure context.

In a similar manner, starting with a BACKCHAIN on the third existential triple in (12) also leads to a failure context. The problem in this example is that restricting search to INTRO and BACKCHAIN forces instances of A_0x to be atomic and no atomic instance leads to a proof. The SETVAR operation, on the other hand, results in a type containing set intersection, which unfolds to conjunction, which further unfolds to a non-atomic type. As we will see in Section 4, without SETVAR, we must use the operation which performs enumeration of types in order to get an instance for A_0x that is not atomic. In general, type enumeration leads to a very large search space. One way to view the SETVAR operation is as a method for controlling type enumeration for theorems in a particular class.

As mentioned, SetVar also serves as a proof search procedure for extensions of hohh and LF. We view either one as a sublanguage of CC^+ by making appropriate restrictions. For example, to restrict the procedure to hohh, we must restrict the types of bound variables in context elements to be types in Church's simple theory of types. In addition, we must place restrictions on the syntax of types that are analogous to the restrictions placed on formulas of higher-order logic in hohh. To describe one of the restrictions, we define the notion of positive and negative occurrences of terms in formulas. If a term A occurs in a base type P in some context Γ, A is said to *occur positively* in P. Term A *occurs positively* (*negatively*) in $\forall x\!:\!P.Q$

of the restrictions on the syntax of formulas is that for every base type $xM_1 \dots M_n$, if this term appears positively in a universal declaration, then x cannot be an existential variable; it must be a universal variable. Similarly, if $xM_1 \dots M_n$ appears negatively in an existential declaration, then x must be a universal variable. In our extension, we relax this restriction and allow x to be an existential variable whenever its type has the form $\tau_1 \to \cdots \to \tau_j \to \tau'_1 \to \cdots \to \tau'_k \to \mathit{Prop}$ for some $j \geq 0$ and $k > 0$, the type $\tau'_1 \to \cdots \to \tau'_k \to \mathit{Prop}$ is a set type, and x only occurs in expressions of the form $\langle x_1, \dots, x_k \rangle \in xz_1 \dots z_j$ where $xz_1 \dots z_j$ is a set variable. In this sublanguage of CC$^+$, the INTRO and BACKCHAIN operations correspond fairly closely to search operations in the λProlog interpreter, while the SETVAR operation handles instantiation of set variables in the extended language. In addition, the SETVAR operation within the context of this extended version of hohh gives a formalization of Bledsoe's procedure in a higher-order logic setting. In contrast, SetVar is described in an adhoc extension to first-order logic in Bledsoe [3].

To use this procedure for proof search in LF, we must extend LF to permit quantification over certain predicates. We permit such quantification in a restricted way, similar to the way it is permitted in the extension of hohh above. In particular, we allow existential quantification over predicate x whenever the following conditions hold: the type of x type has the form $\forall \overline{x_j} : \overline{A_j}.\forall \overline{z_k} : \overline{B_k}.\mathit{Prop}$ for some $j \geq 0$ and $k > 0$; the types $A_1, \dots, A_j$ are any LF types; the types $B_1, \dots, B_k$ are base types in LF (which means that the type $\forall \overline{z_k} : \overline{B_k}.\mathit{Prop}$ is a set type of a particular form); and x only occurs in expressions of the form $\langle x_1, \dots, x_k \rangle \in xz_1 \dots z_j$ where $xz_1 \dots z_j$ is a set variable. In Bledsoe's setting, after instantiating all set variables, the formula becomes a formula of first-order logic. Similarly, in LF with the extension just described, whenever a context has a derivation, it will be the case that after instantiation of existential quantifiers, the result is a valid context in pure LF. For LF, an additional change is needed. Because LF does not permit general quantification over predicates, we cannot use the direct encoding of logical connectives and set operations described in the previous section. Instead, these definitions need to be axiomatized in LF.

3.2 Soundness of the SetVar Search Procedure

We begin by stating and proving some general properties about search contexts, substitution, and normal forms.

Given term P and context Γ, we write $\beta\eta(P, \Gamma)$ (or just $\beta\eta(P)$ when Γ obvious) to denote the normal form of P in Γ if it has one. Similarly, we write $\beta\eta(\Gamma)$ to denote the normal form of context Γ. Let Γ be a valid search context. When applying a series of substitutions to a context or term, it is easy to see that if a normalization is performed after all substitutions are completed, then any intermediate normalization steps have no effect. The following lemma states this fact.

Lemma 1. *Let P be a term, let Γ be a context, and let σ and τ be two substitutions. If $\tau\sigma P$ has a normal form in $\tau\sigma\Gamma$, then $\beta\eta(\tau(\beta\eta(\sigma P))) = \beta\eta(\tau\sigma P)$. Also $\beta\eta(\tau(\beta\eta(\sigma\Gamma))) = \beta\eta(\tau\sigma\Gamma)$.*

The next two lemmas about search contexts follow directly from properties about standard contexts in [10].

Lemma 2. *Let Γ be a valid search context, let Γ' be its normal form, and let P and Q be two terms such that $\overline{\Gamma} \vdash P : Q$. Then Γ' is a valid search context and $\overline{\Gamma'} \vdash P : Q$.*

failure context. Then there is an existential triple $((z_1:A_1,\ldots,z_n:A_n),z,B)$ *in* Γ, $n\geq 0$, *such that* $\forall z_1:A_1\ldots\forall z_n:A_n.B$ *is well-typed in* $\overline{\Gamma}$ *without using the constraints and* B *has the form* $\forall z_{n+1}:A_{n+1}\ldots\forall z_m:A_m.C$ *where* $m\geq n$ *and* C *is atomic and rigid in* $\overline{\Gamma},z_1:A_1,\ldots,z_m:A_m$.

If x is a variable, P is a term, and Γ is a standard context then $[P/x]\Gamma$ denotes the operation of substituting P for all free occurrences of x in constraints and on the right of declarations in Γ. The following property is known to hold for standard contexts in CC and was shown in Dowek [9] to extend to CC$^+$ contexts.

Lemma 4. *Let* M,N,A *be terms and let* $\Gamma,x:B,\Gamma'$ *be a context such that* $\Gamma,x:B,\Gamma'\vdash M:A$ *and* $\Gamma\vdash N:B$. *Then* $\Gamma,[N/x]\Gamma'$ *is a valid context and* $\Gamma,[N/x]\Gamma'\vdash[N/x]M:[N/x]A$.

The next three lemmas are needed to allow us to adapt additional properties in [10] to our setting. Lemmas 5 and 6 provide the necessary correspondence between standard contexts and search contexts. Lemma 7 introduces a new concept needed for our soundness proof.

Lemma 5. *Let* Γ *be a valid search context and let* σ *be a substitution. Then* $\overline{\sigma}\overline{\Gamma}=\overline{\sigma\Gamma}$.

Proof. The proof is by induction on the length of Γ. The theorem clearly holds if Γ is the empty context. Otherwise, Γ has the form Γ',e and we assume that $\overline{\sigma}\overline{\Gamma'}=\overline{\sigma\Gamma'}$.

For the case when e is a universal declaration of the form $x:P$, $\overline{\sigma}\overline{\Gamma}$ is $\overline{\sigma}\overline{\Gamma'},x:\overline{\sigma}P$ and $\overline{\sigma\Gamma}$ is $\overline{\sigma\Gamma'},x:\sigma P$. By the induction hypothesis and the fact that $\overline{\sigma}P=\sigma P$, these two contexts are the same.

For the case when e is an existential triple of the form $((z_1:A_1,\ldots,z_n:A_n),z,B)$ where $n\geq 0$, then if there is a tuple $\langle z,\Delta,M\rangle$ in σ, then $\sigma\Gamma$ is $\sigma\Gamma',\Delta$ and $\langle z,\overline{\Delta},M\rangle$ is in $\overline{\sigma}$. Thus $\overline{\sigma}\overline{\Gamma}$ is $\overline{\sigma}\overline{\Gamma'},\overline{\Delta}$ and $\overline{\sigma\Gamma}$ is $\overline{\sigma\Gamma'},\overline{\Delta}$ which are the same context by a simple application of the induction hypothesis. Otherwise, $\sigma\Gamma$ is $\sigma\Gamma',((z_1:\sigma A_1,\ldots,z_n:\sigma A_n),z,\sigma B)$. In this case $\overline{\sigma}\overline{\Gamma}$ is $\overline{\sigma}\overline{\Gamma'},\exists z:\forall z_1:\overline{\sigma}A_1\ldots\forall z_n:\overline{\sigma}A_n.\overline{\sigma}B$ and $\overline{\sigma\Gamma}$ is $\overline{\sigma\Gamma'},\exists z:\forall z_1:\sigma A_1\ldots\forall z_n:\sigma A_n.\sigma B$ which are again the same context because σ and $\overline{\sigma}$ are the same substitution on terms.

The case when e is a constraint triple is similar to the case for existential triples when the existential variable is not bound by σ. ■

Let Γ be a valid search context. A substitution σ is *well-typed in* Γ if $\sigma\Gamma$ is a valid context, for every tuple $\langle z,\Delta,M\rangle\in\sigma$, either z does not occur in Γ or if it occurs, Γ has the form $\Gamma',((z_1:A_1,\ldots,z_n:A_n),z,B),\Gamma''$ and $\overline{\sigma\Gamma'},\overline{\Delta}\vdash M:\sigma(\forall z_1:A_1\ldots\forall z_n:A_n.B)$ holds. We can assume that all the existential variables introduced in the context argument of tuples in σ are distinct from one another.

Lemma 6. *Let* Γ *be a valid search context and* σ *a substitution. Then* σ *is well-typed in* Γ *if and only if* $\overline{\sigma}$ *is well-typed in* $\overline{\Gamma}$.

Proof. The proof is by induction on the length of Γ. The theorem clearly holds if Γ is the empty context. Otherwise, Γ has the form Γ',e. We must show that σ is well-typed in Γ',e if and only if $\overline{\sigma}$ is well-typed in $\overline{\Gamma'},\overline{e}$. We only show the case for the forward direction when e is an existential triple of the form $((z_1:A_1,\ldots,z_n:A_n),z,B)$ where $n\geq 0$. The other cases are similar, and the proof is easily reversed to get the backward direction.

is well-typed in Γ', so by the induction hypothesis, we know that $\overline{\sigma}$ is well-typed in $\overline{\Gamma'}$. Thus, by definition of well-typed substitution, $\sigma\Gamma'$ is a valid search context and $\overline{\sigma}\,\overline{\Gamma'}$ is a valid standard context.

We first consider the case when z does not occur as the first argument in a tuple in σ. Since σ is well-typed in $\Gamma', ((z_1:A_1,\ldots,z_n:A_n),z,B)$, we know that $\sigma\Gamma', ((z_1:\sigma A_1,\ldots,z_n:\sigma A_n),z,\sigma B)$ is a valid search context. Thus, by definition, $\overline{\sigma\Gamma'}, \exists z : \sigma(\forall z_1 : A_1,\ldots,\forall z_n : A_n.B)$ is a valid standard context. By Lemma 5, this context is the same as $\overline{\sigma}\,\overline{\Gamma'}, \exists z:\sigma(\forall z_1:A_1,\ldots,\forall z_n:A_n.B)$. We must show that $\overline{\sigma}$ is well-typed in $\overline{\Gamma'}, \exists z:\forall z_1:A_1,\ldots,\forall z_n:A_n.B$. This follows if we can show that $\overline{\sigma}\,\overline{\Gamma'}, \exists z:\overline{\sigma}(\forall z_1:A_1,\ldots,\forall z_n:A_n.B)$ is a valid context. This follows from the valid standard context above and the fact that σ and $\overline{\sigma}$ are the same when applied to terms.

If there is a tuple $\langle z,\Delta,M\rangle$ in σ, then from the fact that σ is well-typed in Γ', e, we know that $\sigma\Gamma', \sigma e$ is a valid search context, from which it follows that $\sigma\Gamma', \Delta$ is a valid search context. Thus, $\overline{\sigma\Gamma'}, \overline{\Delta}$ is a valid standard context. We also know that $\overline{\sigma\Gamma'}, \overline{\Delta} \vdash M : \sigma(\forall z_1 : A_1 \ldots \forall z_n : A_n.B)$ holds. The tuple $\langle z,\overline{\Delta},M\rangle$ is in $\overline{\sigma}$, so we must show that $\overline{\sigma}\,\overline{\Gamma'}, \overline{\Delta}$ is a valid context. By Lemma 5, this is the same context as $\overline{\sigma\Gamma'}, \overline{\Delta}$ which we have shown to be valid. We must also show that $\overline{\overline{\sigma}\,\overline{\overline{\Gamma'}}}, \overline{\overline{\Delta}} \vdash M : \overline{\sigma}(\forall z_1 : A_1 \ldots \forall z_n : A_n.B)$ holds. Note that $\overline{\overline{\sigma}\,\overline{\overline{\Gamma'}}}$ is the same as $\overline{\overline{\sigma}}\,\overline{\overline{\overline{\Gamma'}}}$ by Lemma 5, which is the same as $\overline{\sigma}\,\overline{\Gamma'}$, which again by Lemma 5, is the same as $\overline{\sigma\Gamma'}$. Also $\overline{\overline{\Delta}}$ is $\overline{\Delta}$. From these equivalences, and the fact that σ and $\overline{\sigma}$ are the same when applied to terms, the above judgment is equivalent to $\overline{\sigma\Gamma'}, \overline{\Delta} \vdash M : \sigma(\forall z_1:A_1 \ldots \forall z_n:A_n.B)$ which we have shown to hold.
■

We introduce a weaker notion of well-typed substitution restricted to the normal form of a context. A substitution σ is *$\beta\eta$-well-typed in* Γ if $\beta\eta(\sigma\Gamma)$ is a valid context, for every tuple $\langle z,\Delta,M\rangle \in \sigma$, either z does not occur in Γ or if it occurs, Γ has the form $\Gamma', ((z_1:A_1,\ldots,z_n:A_n),z,B),\Gamma''$, both M and $\sigma(\forall z_1 : A_1 \ldots \forall z_n : A_n.B)$ have normal forms in $\beta\eta(\overline{\sigma\Gamma'}, \overline{\Delta})$, and $\beta\eta(\overline{\sigma\Gamma'}, \overline{\Delta}) \vdash \beta\eta(M) : \beta\eta(\sigma(\forall z_1:A_1 \ldots \forall z_n:A_n.B))$ holds.

Lemma 7. *Let Γ be a valid search context and let σ be a substitution. If σ is well-typed in Γ, then σ is $\beta\eta$-well-typed in Γ.*

Proof. This theorem follows directly from the definition of well-typed substitution and Lemma 2.
■

The next four lemmas follow directly from Lemmas 5, 6, and 7, and properties in [10]. We give the proof of Lemma 10 only.

Lemma 8. *Let Γ be a valid search context, σ a substitution, and P and Q two terms such that $\overline{\Gamma} \vdash P : Q$. If σ is well-typed in Γ, then $\sigma\Gamma$ is a valid context and $\overline{\sigma\Gamma} \vdash \sigma P : \sigma Q$. If σ is $\beta\eta$-well-typed in Γ, then $\beta\eta(\sigma\Gamma)$ is a valid context and $\beta\eta(\overline{\sigma\Gamma}) \vdash \beta\eta(\sigma P) : \beta\eta(\sigma Q)$.*

The *composition* of two substitutions σ and τ, denoted $\tau \circ \sigma$, is the union of the set of triples $\langle z,\tau\Delta,\tau M\rangle$ such that $\langle z,\Delta,M\rangle \in \sigma$, and the set of triples $\langle z,\Delta,M\rangle$ such that $\langle z,\Delta,M\rangle \in \tau$ and z is does not occur as the first element of a triple in σ.

Lemma 9. *Let σ and τ be two substitutions and let Γ be a search context. Then $(\tau \circ \sigma)\Gamma = \tau\sigma\Gamma$.*

Lemma 10. *Let Γ be a valid search context and let σ and τ be two substitutions.*

2. If σ is $\beta\eta$-well-typed in Γ and τ is $\beta\eta$-well-typed in the normal form of $\sigma\Gamma$, then $\tau \circ \sigma$ is $\beta\eta$-well-typed in Γ.

Proof. Assume that σ is well-typed in Γ and τ is well-typed in $\sigma\Gamma$. Then $\tau\sigma\Gamma$ is a valid context, and so by the equivalence of Lemma 9, $(\tau \circ \sigma)\Gamma$ is a valid context. Every tuple in $\tau \circ \sigma$ either comes from σ or τ. We first consider tuples from σ. Let $\langle z, \Delta, M\rangle$ be such a tuple. Then $\langle z, \tau\Delta, \tau M\rangle$ is in $\tau \circ \sigma$. If Γ has the form $\Gamma', ((z_1 : A_1, \ldots, z_n : A_n), z, B), \Gamma''$, we must show that

$$\overline{(\tau \circ \sigma)\Gamma'}, \overline{\tau\Delta} \vdash \tau M : (\tau \circ \sigma)(\forall z_1 : A_1 \ldots \forall z_n : A_n . B). \tag{1}$$

Since σ is well-typed in Γ, we know that

$$\overline{\sigma\Gamma'}, \overline{\Delta} \vdash M : \sigma(\forall z_1 : A_1 \ldots \forall z_n : A_n . B). \tag{2}$$

We know that τ is well-typed in $\sigma\Gamma$ and by definition of substitution, $\sigma\Gamma', \Delta$ is a subcontext of $\sigma\Gamma$. Thus, τ is well-typed in $\sigma\Gamma', \Delta$. So from (2) and Lemma 8, we know that

$$\overline{\tau\sigma\Gamma'}, \overline{\tau\Delta} \vdash \tau M : \tau\sigma(\forall z_1 : A_1 \ldots \forall z_n : A_n . B)$$

which by Lemma 9 is equivalent to (1) and we have our result. We now consider tuples from τ. Let $\langle z, \Delta, M\rangle$ be such a tuple. By definition of composition, we know that this tuple is in $\tau \circ \sigma$ and that z is not bound by σ. If Γ has the form $\Gamma', ((z_1 : A_1, \ldots, z_n : A_n), z, B), \Gamma''$, we must show that

$$\overline{(\tau \circ \sigma)\Gamma'}, \overline{\Delta} \vdash M : (\tau \circ \sigma)(\forall z_1 : A_1 \ldots \forall z_n : A_n . B). \tag{3}$$

Since z is not bound by σ, we know that $\sigma\Gamma$ has the form $\sigma\Gamma', ((z_1 : \sigma A_1, \ldots, z_n : \sigma A_n), z, \sigma B), \sigma\Gamma''$. Since τ is well-typed in $\sigma\Gamma$, we know that $\overline{\tau\sigma\Gamma'}, \overline{\Delta} \vdash M : \tau\sigma(\forall z_1 : A_1 \ldots \forall z_n : A_n . B)$, which by Lemma 9 is equivalent to (3) and we have our result.

For the case when σ is $\beta\eta$-well-typed in Γ and τ is $\beta\eta$-well-typed in the normal form of $\sigma\Gamma$, the proof is similar and also relies on Lemmas 1 and 5. ■

Given valid search context Γ, a substitution σ is said to be a *solution* to Γ if σ is $\beta\eta$-well-typed in Γ and $\beta\eta(\sigma\Gamma)$ is a success context. A solution is *normal* if it binds exactly the existential variables of Γ and for every tuple $\langle z, \Delta, M\rangle$ such that Γ has the form $\Gamma', ((z_1 : A_1, \ldots, z_n : A_n), z, B), \Gamma''$, we have that Δ is empty and M is normal in $\overline{\sigma\Gamma'}$. For an arbitrary solution σ to a context Γ, we obtain the *normal form of* σ from σ as follows: remove all tuples $\langle z, \Delta, M\rangle$ such that z is not an existential variable in Γ; for all other tuples $\langle z, \Delta, M\rangle \in \sigma$ such that Γ has the form $\Gamma', ((z_1 : A_1, \ldots, z_n : A_n), z, B), \Gamma''$, replace this tuple with $\langle z, \langle\rangle, M'\rangle$ where M' is the normal form of M in $\overline{\sigma\Gamma'}$.

Lemma 11. *Let Γ be a valid search context and let σ be a solution to Γ. Let σ' be the normal form of σ. Then σ' is a normal solution to Γ.*

The remaining lemmas and their proofs follow fairly closely the proof of soundness in Dowek [10]. The main differences are that we must prove additional cases for the SETVAR operation and the cases for INTRO and BACKCHAIN are slightly modified because of the use of search contexts.

Let $\{\langle z, \Delta, M\rangle\}$ be the result of applying a search operation to Γ. Then Γ', Δ is a valid search context.

Proof. Let Δ' be the single item context $((z_1 : A_1, \ldots, z_n : A_n), z, C)$. Since Γ is a valid search context, Γ' and Γ', Δ' are valid search contexts, and thus $\overline{\Gamma'}$ and $\overline{\Gamma'}, \overline{\Delta'}$ are valid standard contexts. To show that Γ', Δ is valid, we need to show that $\overline{\Gamma'}, \overline{\Delta}$ is a valid standard context.

For the SETVAR case, Δ is empty and we have our result.

For the INTRO case, C has the form $\forall x : A.B$, M is some new variable z' and Δ is $((z_1 : A_1, \ldots, z_n : A_n, x : A), z', B)$. Since $\overline{\Delta'}$ is the same as $\overline{\Delta}$ up to renaming of the existential variable and $\overline{\Gamma'}, \overline{\Delta'}$ is valid, we have our result.

For the BACKCHAIN case, there is a declaration $w : \forall y_1 : Q_1 \ldots \forall y_q : Q_q.B$ with $q \geq 0$ which either occurs in Γ' or w is one of $z_1, \ldots, z_n$. $\overline{\Delta}$ is

$$\begin{array}{l}
\exists h_1 : \forall \overline{z}_n : \overline{A}_n.Q_1, \\
\exists h_2 : \forall \overline{z}_n : \overline{A}_n.[h_1 z_1 \ldots z_n / y_1]Q_2, \\
\quad \vdots \\
\exists h_q : \forall \overline{z}_n : \overline{A}_n.[h_1 z_1 \ldots z_n / y_1, \ldots, h_{q-1} z_1 \ldots z_n / y_{q-1}]Q_q, \\
\forall \overline{z}_n : \overline{A}_n.[h_1 z_1 \ldots z_n / y_1, \ldots, h_q z_1 \ldots z_n / y_q]B = \forall \overline{z}_n : \overline{A}_n.C.
\end{array}$$

From the definition of BACKCHAIN, we know that the following hold

$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n \vdash C : s \tag{1}$$

$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n \vdash \forall y_1 : Q_1 \ldots \forall y_q : Q_q.B : s$$

where s is *Prop* or *Type*. Thus, for $i = 1, \ldots, n$, there is a sort s_i, and for $j = 1, \ldots, q$, there is a sort s'_j such that the following hold.

$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_{i-1} : A_{i-1} \vdash A_i : s_i \tag{2}$$

$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n, y_1 : Q_1, \ldots, y_{j-1} : Q_{j-1} \vdash Q_j : s'_j \tag{3}$$

$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n, y_1 : Q_1, \ldots, y_q : Q_q \vdash B : s \tag{4}$$

For $i = 1, \ldots, q$, let Δ_i be the context containing the first i elements of $\overline{\Delta}$. We prove by induction on q that $\overline{\Gamma'}, \Delta_q$ is valid. If q is 0, Δ_q is empty and we are done. Otherwise assume that $\overline{\Gamma'}, \Delta_{q-1}$ is valid. Since $\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n$ is valid, by thinning we know that $\overline{\Gamma'}, \Delta_{q-1}, z_1 : A_1, \ldots, z_n : A_n$ is valid. Thus, for $i = 1, \ldots, q-1$, by Q-INIT we have that

$$\overline{\Gamma'}, \Delta_{q-1}, z_1 : A_1, \ldots, z_n : A_n \vdash h_i : \forall \overline{z}_n : \overline{A}_n.[h_1 z_1 \ldots z_n / y_1, \ldots, h_{i-1} z_1 \ldots z_n / y_{i-1}]Q_i. \tag{5}$$

From (5), by repeated applications of APP, for $i = 1, \ldots, q-1$, we get

$$\overline{\Gamma'}, \Delta_{q-1}, z_1 : A_1, \ldots, z_n : A_n \vdash h_i z_1 \ldots z_n : [h_1 z_1 \ldots z_n / y_1, \ldots, h_{i-1} z_1 \ldots z_n / y_{i-1}]Q_i. \tag{6}$$

From (2) with $i = 1, \ldots, n$, (3), and thinning, we get

$$\overline{\Gamma'}, \Delta_{q-1}, z_1 : A_1, \ldots, z_{i-1} : A_{i-1} \vdash A_i : s_i \tag{7}$$

$$\overline{\Gamma'}, \Delta_{q-1}, z_1 : A_1, \ldots, z_n : A_n, y_1 : Q_1, \ldots, y_{q-1} : Q_{q-1} \vdash Q_q : s'_q \tag{8}$$

From (6) with $i = 1$, (8), and Lemma 4, we obtain:

$$\begin{array}{l}
\overline{\Gamma'}, \Delta_{q-1}, z_1 : A_1, \ldots, z_n : A_n, \\
\quad y_2 : [h_1 z_1 \ldots z_n / y_1]Q_2, \ldots, y_{q-1} : [h_1 z_1 \ldots z_n / y_1]Q_{q-1} \vdash [h_1 z_1 \ldots z_n / y_1]Q_q : s'_q
\end{array}$$

$$\overline{\Gamma'}, \Delta_{q-1}, z_1 : A_1, \ldots, z_n : A_n \vdash [h_1 z_1 \ldots z_n / y_1, \ldots, h_{q-1} z_1 \ldots z_n / y_{q-1}] Q_q : s'_q. \tag{9}$$

From (9), (7), and repeated applications of PROD, we can conclude

$$\overline{\Gamma'}, \Delta_{q-1} \vdash \forall \overline{z}_n : \overline{A}_n [h_1 z_1 \ldots z_n / y_1, \ldots, h_{q-1} z_1 \ldots z_n / y_{q-1}] Q_q : s'_q$$

from which we can conclude by an application of Q-INTRO that $\overline{\Gamma'}, \Delta_q$ is valid.

It remains to show that the constraint is well-typed in $\overline{\Gamma'}, \Delta_q$. By repeated applications of PROD from (1) and (2), it follows that

$$\overline{\Gamma'} \vdash \forall \overline{z}_n : \overline{A}_n . C : s. \tag{10}$$

By thinning from (2), (4), and (10), the following hold.

$$\overline{\Gamma'}, \Delta_q, z_1 : A_1, \ldots, z_{i-1} : A_{i-1} \vdash A_i : s_i \tag{11}$$

$$\overline{\Gamma'}, \Delta_q \vdash \forall \overline{z}_n : \overline{A}_n . C : s \tag{12}$$

$$\overline{\Gamma'}, \Delta_q, z_1 : A_1, \ldots, z_n : A_n, y_1 : Q_1, \ldots, y_q : Q_q \vdash B : s \tag{13}$$

Since $\overline{\Gamma'}, \Delta_q$ is valid, we now know that (6) holds for $i = 1, \ldots, q$ with Δ_q replacing Δ_{q-1}. Thus by repeated applications of Lemma 4 from (13) using this new version of (6) with $i = 1, \ldots, q$, we obtain the following.

$$\overline{\Gamma'}, \Delta_q, z_1 : A_1, \ldots, z_n : A_n \vdash [h_1 z_1 \ldots z_n / y_1, \ldots, h_q z_1 \ldots z_n / y_q] B : s \tag{14}$$

By repeated applications of PROD from (11) and (14), followed by a single application of EQ-INTRO using (12), we get the desired result. ■

Lemma 13. *Let Γ be a normal valid search context of the form $\Gamma', ((z_1 : A_1, \ldots, z_n : A_n), z, C), \Gamma''$. Let $\{\langle z, \Delta, M \rangle\}$ be the result of applying a search operation to Γ. Then $\overline{\Gamma'}, \overline{\Delta} \vdash M : \forall z_1 : A_1 \ldots \forall z_n : A_n . C$ holds.*

Proof. For the SETVAR case, C is a set type of the form $\forall x_1 : C_1 \ldots \forall x_p : C_p . Prop$ with $p > 0$, $\overline{\Delta}$ is empty, and M has the form $\lambda \overline{z}_n : \overline{A}_n . Q'_1 \cap \cdots \cap Q'_q$ where $q > 0$ and for $i = 1, \ldots, q$, Q'_i is obtained from some formula P_i by an application of rule 1,2,3,4, or 5 followed by 0 or more applications of rule 6 of Fig. 5. We must show that $\overline{\Gamma'} \vdash \lambda \overline{z}_n : \overline{A}_n . Q'_1 \cap \cdots \cap Q'_q : \forall \overline{z}_n : \overline{A}_n . \forall \overline{x}_p : \overline{C}_p . Prop$ holds. This holds if by applications of ABS and the definition of $\cap$, for $i = 1, \ldots, q$, the following holds.

$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n \vdash Q'_i : \forall \overline{x}_p : \overline{C}_p . Prop.$$

(It is straightforward to show that the left premises of this series of applications of ABS hold, and also that if the above judgments hold then $\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n \vdash Q'_1 \cap \cdots \cap Q'_q : \forall \overline{x}_p : \overline{C}_p . Prop$ holds.) For each i, we proceed by induction on the number k of applications of rule 6. In the case where $k = 0$, then Q'_i was obtained from P_i by a single application of one of the rules 1,2,3,4, or 5. Because Γ is valid, we know that $\overline{\Gamma'} \vdash z : \forall \overline{z}_n : \overline{A}_n . \forall \overline{x}_p : \overline{C}_p . Prop$ holds and that the context $\Gamma', z_1 : A_1, \ldots, z_n : A_n$ is valid. By thinning, we get $\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n \vdash z : \forall \overline{z}_n : \overline{A}_n . \forall \overline{x}_p : \overline{C}_p . Prop$ and by repeated applications of APP, we can conclude:

$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n \vdash z z_1 \ldots z_n : \forall x_1 : C_1 \ldots \forall x_p : C_p . Prop. \tag{1}$$

the definition of the SETVAR operation and the provisos in Fig. 5 can be used to show that the cases for rules 1,3,4, and 5 hold.

If Q'_i was obtained from rule 2, then from the fact that the provisos hold, there is some j, r with $0 \leq j < p$ and $j + r = p$ such that (1) can be rewritten as

$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n \vdash zz_1 \ldots z_n : \forall x_1 : C_1 \ldots \forall x_j : C_j . C_{j+1} \to \cdots \to C_{j+r} \to \mathit{Prop}. \quad (2)$$

P_i has the form $\langle x_1, \ldots, x_j, f_1 x_1 \ldots x_{p'}, \ldots, f_r x_1 \ldots x_{p'} \rangle \in zz_1 \ldots z_n \to P' x_1 \ldots x_{p'}$ for some p' such that $p' > j$ and Q'_i has the form

$$\begin{array}{l} \{\langle x_1, \ldots, x_j, w_1, \ldots, w_r \rangle \mid \\ \quad \forall x_{j+1} : D_{j+1} \ldots \forall x_{p'} : D_{p'} . w_1 =_{C_{j+1}} f_1 x_1 \ldots x_{p'} \to \cdots \to w_r =_{C_{j+r}} f_r x_1 \ldots x_{p'} \to P' x_1 \ldots x_{p'} \} \end{array}$$

for some terms $D_{j+1}, \ldots, D_{p'}$. We can prove that this term has type $\forall x_1 : C_1 \ldots \forall x_j : C_j . C_{j+1} \to \cdots \to C_{j+r} \to \mathit{Prop}$ in context $\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n$ if (by unfolding of the set notation and applications of ABS and PROD in a backward direction) we can prove that the following judgment holds. (Again, the left premises of the applications of ABS and PROD follow easily.)

$$\begin{array}{l} \overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n, x_1 : C_1, \ldots, x_j : C_j, w_1 : C_{j+1}, \ldots, w_r : C_{j+r}, x_{j+1} : D_{j+1}, \ldots, x_{p'} : D_{p'} \\ \vdash w_1 =_{C_{j+1}} f_1 x_1 \ldots x_{p'} \to \cdots \to w_r =_{C_{j+r}} f_r x_1 \ldots x_{p'} \to P' x_1 \ldots x_{p'} : \mathit{Prop} \end{array}$$

This follows directly from the types of $=_{C_i}$ for $i = j+1, \ldots, j+r$, the types given in the provisos of $P', f_1, \ldots, f_r$, and applications of PROD.

For the induction case, when $k > 0$, Q'_i has the form $\{\langle y_1, \ldots, y_{q'} \rangle \mid (\exists\ \lambda w : C' . Q')\}$ and we must show that

$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n \vdash \{\langle y_1, \ldots, y_{q'} \rangle \mid (\exists\ \lambda w : C' . Q')\} : \forall x_1 : C_1 \ldots \forall x_p : C_p . \mathit{Prop} \quad (3)$$

holds under the assumption that

$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n \vdash \{\langle y_1, \ldots, y_{q'} \rangle \mid Q'\} : \forall x_1 : C_1 \ldots \forall x_p : C_p . \mathit{Prop} \quad (4)$$

holds. Note that for this judgment to be derivable, it must be the case that $q' \leq p$. Variables can be renamed so that $y_1, \ldots, y_{q'}$ are the same variables as $x_1, \ldots, x_q$. Then (3) follows from (4), the type of $\exists$, and the fact that according to the provisos in Fig. 5, w does not occur free elsewhere in $\overline{\Gamma'}$.

For the INTRO case, C has the form $\forall x : A . B$, M is some new variable z' and $\overline{\Delta}$ is $\exists z' : \forall \overline{z}_n : \overline{A}_n . \forall x : A . B$. Then directly by Q-INIT, $\overline{\Gamma'}, \overline{\Delta} \vdash z' : \forall \overline{z}_n : \overline{A}_n . \forall x : A . B$ holds.

For the BACKCHAIN case, there is a declaration $w : \forall y_1 : Q_1 \ldots \forall y_q : Q_q . B$ with $q \geq 0$ which either occurs in Γ' or w is one of $z_1, \ldots, z_n$. The term M is $\lambda \overline{z}_n : \overline{A}_n . w(h_1 z_1 \ldots z_n) \ldots (h_q z_1 \ldots z_n)$ and $\overline{\Delta}$ is

$$\begin{array}{l} \exists h_1 : \forall \overline{z}_n : \overline{A}_n . Q_1, \\ \exists h_2 : \forall \overline{z}_n : \overline{A}_n . [h_1 z_1 \ldots z_n / y_1] Q_2, \\ \quad \vdots \\ \exists h_q : \forall \overline{z}_n : \overline{A}_n . [h_1 z_1 \ldots z_n / y_1, \ldots, h_{q-1} z_1 \ldots z_n / y_{q-1}] Q_q, \\ \forall \overline{z}_n : \overline{A}_n . [h_1 z_1 \ldots z_n / y_1, \ldots, h_q z_1 \ldots z_n / y_q] B = \forall \overline{z}_n : \overline{A}_n . C. \end{array}$$

We must show that

$$\overline{\Gamma'}, \overline{\Delta} \vdash \lambda \overline{z}_n : \overline{A}_n . w(h_1 z_1 \ldots z_n) \ldots (h_q z_1 \ldots z_n) : \forall \overline{z}_n : \overline{A}_n . C \quad (5)$$

$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n \vdash C : s$$
$$\overline{\Gamma'}, z_1 : A_1, \ldots, z_n : A_n \vdash w : \forall y_1 : Q_1 \ldots \forall y_q : Q_q.B$$

holds where s is either *Prop* or *Type*. By Lemma 12, we know that $\overline{\Gamma'}, \overline{\Delta}$ is a valid context, so by thinning, the following hold.

$$\overline{\Gamma'}, \overline{\Delta}, z_1 : A_1, \ldots, z_n : A_n \vdash C : s \tag{6}$$
$$\overline{\Gamma'}, \overline{\Delta}, z_1 : A_1, \ldots, z_n : A_n \vdash w : \forall y_1 : Q_1 \ldots \forall y_q : Q_q.B \tag{7}$$

From (6), by applications of PROD (where the left premises are easy to prove as before), we can conclude
$$\overline{\Gamma'}, \overline{\Delta} \vdash \forall \overline{z}_n : \overline{A}_n C : s. \tag{8}$$
Since $\overline{\Gamma}, \overline{\Delta}$ is valid, we also know that both sides of the constraint in $\overline{\Delta}$ have the same type. Thus, from (8), we know

$$\overline{\Gamma'}, \overline{\Delta} \vdash \forall \overline{z}_n : \overline{A}_n.[h_1 z_1 \ldots z_n / y_1, \ldots, h_q z_1 \ldots z_n / y_q]B : s \tag{9}$$

holds. Also, for $i = 1, \ldots q$, the following hold.

$$\overline{\Gamma'}, \overline{\Delta}, z_1 : A_1, \ldots, z_n : A_n \vdash h_i z_1 \ldots z_n : [h_1 z_1 \ldots z_n / y_1, \ldots, h_{i-1} z_1 \ldots z_n / y_{i-1}]Q_i \tag{10}$$

By repeated applications of APP from (7) and (10)

$$\overline{\Gamma'}, \overline{\Delta}, z_1 : A_1, \ldots, z_n : A_n \vdash w(h_1 z_1 \ldots z_n) \ldots (h_q z_1 \ldots z_n) : [h_1 z_1 \ldots z_n / y_1, \ldots, h_q z_1 \ldots z_n / y_q]B$$

holds, and by repeated applications of ABS where the left premises are easy to prove as before

$$\begin{aligned} \overline{\Gamma'}, \overline{\Delta} \quad \vdash \quad & \lambda \overline{z}_n : \overline{A}_n . w(h_1 z_1 \ldots z_n) \ldots (h_q z_1 \ldots z_n) : \\ & \forall \overline{z}_n : \overline{A}_n.[h_1 z_1 \ldots z_n / y_1, \ldots, h_q z_1 \ldots z_n / y_q]B \end{aligned} \tag{11}$$

holds. Thus by an application of CONV from (8), (9), (11), and the constraint in $\overline{\Delta}$, we can conclude that the desired result (5) holds. ■

Lemma 14. *Let* Γ *be a normal valid search context of the form* $\Gamma', ((z_1 : A_1, \ldots, z_n : A_n), z, C), \Gamma''$. *Let* $\{\langle z, \Delta, M \rangle\}$ *be the result of applying a search operation to* Γ. *Let* σ *be the substitution containing the single tuple* $\langle z, \Delta, M \rangle$. *Then* σ *is well-typed in* Γ.

Proof. The proof is by induction on the number of elements in Γ''. Note that $\sigma\Gamma'$ is just Γ'. Let Δ' be the context containing the single element $((z_1 : A_1, \ldots, z_n : A_n), z, C)$.

For the base case, when Γ'' is empty, we have to show that $\sigma(\Gamma', \Delta')$ is valid and that $\overline{\sigma\Gamma'}, \overline{\Delta} \vdash M : \sigma(\forall \overline{z}_n : \overline{A}_n.C)$ holds. By the definition of substitution, $\sigma(\Gamma', \Delta')$ is simply $\sigma\Gamma', \Delta$. Since $\sigma\Gamma'$ is Γ', we have to show that Γ', Δ is valid. This follows by Lemma 12. Thus $\sigma(\Gamma', \Delta')$ is valid. Since $\sigma\Gamma'$ is Γ', it is also the case that $\overline{\sigma\Gamma'}$ is $\overline{\Gamma'}$. Also, since Γ is valid, z does not occur free in $A_1, \ldots, A_n, C$, so $\sigma(\forall \overline{z}_n : \overline{A}_n.C)$ is $\forall \overline{z}_n : \overline{A}_n.C$. Thus, we have to show $\overline{\Gamma'}, \overline{\Delta} \vdash M : \forall \overline{z}_n : \overline{A}_n.C$. This follows by Lemma 13.

If Γ'' is non-empty, it has the form Δ'', e. Thus, $\sigma\Gamma$ is $\Gamma', \Delta, \sigma\Delta'', \sigma(e)$. To show that σ is well-typed in Γ, we must show that $\Gamma', \Delta, \sigma\Delta'', \sigma(e)$ is a valid search context, or equivalently is well-typed in $\Gamma', \Delta', \Delta''$ and thus $\Gamma', \Delta, \sigma\Delta''$ is a valid context.

We show the case when e is an existential triple of the form $((y_1 : B_1, \ldots, y_m : B_m), y, D)$ where $m \geq 0$. The others are similar. Note that $\overline{\Gamma}$ is $\overline{\Gamma'}, \overline{\Delta'}, \overline{\Delta''}, \exists y : \forall \overline{y}_m : \overline{B}_m.D$ and that $\overline{\sigma\Gamma}$ is $\overline{\Gamma'}, \overline{\Delta}, \overline{\sigma\Delta''}, \exists y : \sigma(\forall \overline{y}_m : \overline{B}_m.D)$. Note that the Q-INTRO rule was the last rule in a proof that $\overline{\Gamma}$ is valid and thus $\overline{\Gamma'}, \overline{\Delta'}, \overline{\Delta''} \vdash \forall \overline{y}_m : \overline{B}_m.D : s$ holds where s is *Prop* or *Type*. Since σ is well-typed in $\Gamma', \Delta', \Delta''$, by Lemma 8 we know that $\overline{\Gamma'}, \overline{\Delta}, \overline{\sigma\Delta''} \vdash \sigma(\forall \overline{y}_m : \overline{B}_m.D) : s$ and thus the standard context $\overline{\Gamma'}, \overline{\Delta}, \overline{\sigma\Delta''}, \exists y : \sigma(\forall \overline{y}_m : \overline{B}_m.D)$ is valid, and hence so is the corresponding search context $\Gamma', \Delta, \sigma\Delta'', ((y_1 : \sigma B_1, \ldots, y_m : \sigma B_m), y, \sigma D)$. ■

Let Γ be a normal valid search context and let $\sigma_1, \ldots, \sigma_n$ be a derivation of Γ. The normal form of $\sigma_n \circ \cdots \circ \sigma_1$ is called the *substitution denoted by the derivation* $\sigma_1, \ldots, \sigma_n$.

Lemma 15. *Let Γ be a valid search context and let $\sigma_1, \ldots, \sigma_n$ be a derivation of Γ. Then the substitution denoted by the derivation $\sigma_1, \ldots, \sigma_n$ is a solution to Γ.*

Proof. We first prove that $\sigma_n \circ \cdots \circ \sigma_1$ is $\beta\eta$-well-typed in Γ by induction on n. If n is 0, then $\sigma_n \circ \cdots \circ \sigma_1$ is empty and we only need to show that $\beta\eta(\sigma_n \circ \cdots \circ \sigma_1\Gamma)$ is valid. Note that $\sigma_n \circ \cdots \circ \sigma_1\Gamma$ is Γ. Since Γ is valid, by Lemma 2 we can conclude that its normal form is valid. For the induction case, we assume that $\sigma_{n-1} \circ \cdots \circ \sigma_1$ is $\beta\eta$-well-typed in Γ. Thus $\beta\eta(\sigma_{n-1} \circ \cdots \circ \sigma_1\Gamma)$ is valid. Since σ_n is the result of applying one of the search operations to this context, by Lemma 14, we know that σ_n is well-typed in $\beta\eta(\sigma_{n-1} \circ \cdots \circ \sigma_1\Gamma)$, and by Lemma 7, it is $\beta\eta$-well-typed in $\beta\eta(\sigma_{n-1} \circ \cdots \circ \sigma_1\Gamma)$. Since $\sigma_{n-1} \circ \cdots \circ \sigma_1$ is $\beta\eta$-well-typed in Γ and σ_n is $\beta\eta$-well-typed in $\beta\eta(\sigma_{n-1} \circ \cdots \circ \sigma_1\Gamma)$, by Lemma 10, $\sigma_n \circ \cdots \circ \sigma_1$ is $\beta\eta$-well-typed in Γ.

By the definition of derivation, $\beta\eta(\sigma_n \ldots \sigma_1\Gamma)$ is a success context. By Lemma 9, this is the same context as $\beta\eta(\sigma_n \circ \cdots \circ \sigma_1\Gamma)$. Since $\beta\eta(\sigma_n \circ \cdots \circ \sigma_1\Gamma)$ is a success context and $\sigma_n \circ \cdots \circ \sigma_1$ is $\beta\eta$-well-typed in Γ, we can conclude that $\sigma_n \circ \cdots \circ \sigma_1$ is a solution to Γ. By Lemma 11, its normal form is also a solution.

Theorem 16. *(Soundness) Let Γ be a normal valid CC context (without existential variables or constraints) and let A be a normal term of type Prop or Type in Γ. Let Γ' be the search context $\Gamma, (\langle\rangle, z, A)$. If there exists a derivation of Γ', then there exists a term M such that $\Gamma \vdash M : A$ holds in CC.*

Proof. Let σ be the substitution denoted by a derivation of Γ'. Since σ is normal, it contains a single tuple of the form $\langle z, \langle\rangle, M\rangle$ for some term M in normal form. By Lemma 15, σ is a solution, and thus by definition it is $\beta\eta$-well-typed in Γ'. By definition of $\beta\eta$-well-typed, we know that $\beta\eta(\overline{\sigma\Gamma}) \vdash \beta\eta(M) : \beta\eta(\sigma A)$ holds. Note that $\sigma\Gamma$ is Γ and recall that Γ is normal. Thus $\beta\eta(\overline{\sigma\Gamma})$ is $\overline{\Gamma}$. Since Γ contains no existential triples or constraint triples, $\overline{\Gamma}$ is Γ. Also, since M is normal $\beta\eta(M)$ is M. In addition, since Γ is valid and A is well-typed in Γ, we know that z does not occur in A and thus σA is A. Thus, since A is normal, we have that $\beta\eta(\sigma A)$ is A. So the above judgment is equivalent to $\Gamma \vdash M : A$ and we have our result.

3.3 Maximal Solutions for Set Variables

Let Γ be a normal valid search context of the form $\Gamma', (\Phi, z, A), \Gamma''$ such that Γ' does not contain any existential triples, Φ has the form $z_1 : A_1, \ldots, z_n : A_n$ for some $n \geq 0$, and A is a set type in Γ', Φ. Let σ be a substitution and M a term such that σ contains the single tuple normal form of $\sigma\Gamma$ has a solution and for any substitution σ' containing a single tuple of the form $\langle A, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n.N\rangle$, it is the case that whenever the following hold:

1. σ' is well-typed in Γ;
2. the normal form of $\sigma'\Gamma$ has a solution;
3. there is a term P such that $\overline{\Gamma'}, \Phi \vdash P : M \subseteq N$ holds;

then there is always a term Q such that $\overline{\Gamma'}, \Phi \vdash Q : M =_S N$ holds. Note that it is built into this definition that $\overline{\Gamma'}, \Phi \vdash M : A$ and $\overline{\Gamma'}, \Phi \vdash N : A$ hold.

Theorems 17-21 justify the maximal solutions given in Fig. 5, while Theorem 22 justifies taking the intersection of maximal solutions of different occurrences of a set variable as done in the SETVAR operation in Sect. 3. The proofs are similar to the proofs in Bledsoe [3] but require extensions for our setting. We give the proof of Theorem 17 for illustration and sketch the others.

Theorem 17. *Let Φ and Φ' be contexts of the form $z_1 : A_1, \ldots, z_n : A_n$ and $x_1 : C_1, \ldots, x_p : C_p$, respectively, where $n \geq 0$ and $p > 0$. Let Γ be a normal valid search context of the form*

$$\Gamma', (\Phi, z, A), ((\Phi, \Phi'), h, \langle x_1, \ldots, x_p\rangle \in zz_1 \ldots z_n \to Px_1 \ldots x_p)$$

such that Γ' does not contain any existential or constraint triples, A is a set type of the form $\forall x_1 : C_1 \ldots \forall x_p : C_p.Prop$, the judgment $\overline{\Gamma'}, \Phi \vdash P : A$ holds, and the terms in Φ' contain no free occurrences of z. Then $\{\langle x_1, \ldots, x_p\rangle \mid Px_1 \ldots x_p\}$ is a maximal solution for $zz_1 \ldots z_n$ in Γ.

Proof. Let σ and τ be the substitutions containing the single tuples

$$\langle z, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n.\{\langle x_1, \ldots, x_p\rangle \mid Px_1 \ldots x_p\}\rangle \text{ and } \langle h, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n.\lambda\overline{x}_p : \overline{C}_p.\lambda x : Px_1 \ldots x_p.x\rangle,$$

respectively. We first show that τ is a solution to the normal form of $\sigma\Gamma$.

First note that $\beta\eta(\tau(\beta\eta(\sigma\Gamma)))$ is $\beta\eta(\Gamma')$, which is just Γ' since Γ (and therefore Γ') is normal. Γ' is a success context since it is valid and contains no existential or constraint triples. It remains to show that τ is $\beta\eta$-well-typed in $\beta\eta(\sigma\Gamma)$. Note that $\beta\eta(\sigma\Gamma)$ is

$$\beta\eta(\Gamma', ((\Phi, \Phi'), h, \langle x_1, \ldots, x_p\rangle \in \{\langle x_1, \ldots, x_p\rangle \mid Px_1 \ldots x_p\} \to Px_1 \ldots x_p))$$

which after expanding definitions and normalizing, results in a context of the form

$$\Gamma', ((\Phi, \Phi'), h, Px_1 \ldots x_p \to Px_1 \ldots x_p).$$

We must show that

$$\begin{aligned} \beta\eta(\tau\overline{\Gamma'}) \quad \vdash \quad & \beta\eta(\lambda\overline{z}_n : \overline{A}_n.\lambda\overline{x}_p : \overline{C}_p.\lambda x : Px_1 \ldots x_p.x) : \\ & \beta\eta(\forall\overline{z}_n : \overline{A}_n.\forall\overline{x}_p : \overline{C}_p.Px_1 \ldots x_p \to Px_1 \ldots x_p) \end{aligned} \tag{1}$$

holds. It is straightforward to construct a proof of

$$\Gamma' \vdash (\lambda\overline{z}_n : \overline{A}_n.\lambda\overline{x}_p : \overline{C}_p.\lambda x : Px_1 \ldots x_p.x) : (\forall\overline{z}_n : \overline{A}_n.\forall\overline{x}_p : \overline{C}_p.Px_1 \ldots x_p \to Px_1 \ldots x_p) \tag{2}$$

and the fact that $\tau\sigma\Gamma'$ is Γ' which we know to be in normal form and valid, we have that τ is $\beta\eta$-well-typed in the normal form of $\sigma\Gamma$. Since Γ' is also a success context, we have that τ is a solution to the normal form of $\sigma\Gamma$.

We must now show that $\{\langle x_1,\ldots,x_p\rangle \mid Px_1\ldots x_p\}$ is maximal. Assume that there are terms N, P' and substitution σ' containing the single tuple $\langle z, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n.N\rangle$ such that σ' is well-typed in Γ, the normal form of $\sigma'\Gamma$ has a solution, and the judgment

$$\overline{\Gamma'}, \Phi \vdash P' : \{\langle x_1,\ldots,x_p\rangle \mid Px_1\ldots x_p\} \subseteq N \tag{3}$$

holds. We must show that there is a term Q such that

$$\overline{\Gamma'}, \Phi \vdash Q : \{\langle x_1,\ldots,x_p\rangle \mid Px_1\ldots x_p\} =_S N$$

or equivalently

$$\overline{\Gamma'}, \Phi \vdash Q : (\{\langle x_1,\ldots,x_p\rangle \mid Px_1\ldots x_p\} \subseteq N) \wedge (N \subseteq \{\langle x_1,\ldots,x_p\rangle \mid Px_1\ldots x_p\}) \tag{4}$$

holds. Note that z does not occur free in Φ, Φ', or P. Thus $\beta\eta(\sigma'\Gamma)$ has the form

$$\Gamma', ((\Phi,\Phi'), h, \langle x_1,\ldots,x_p\rangle \in N \to Px_1\ldots x_p).$$

Since $\beta\eta(\sigma'\Gamma)$ has a solution, by Lemma 11, we know it has a normal solution, say τ', containing a single tuple of the form $\langle h, \langle\rangle, Q'\rangle$ where Q' is a term in normal form. Since τ' is a solution, we know that it is $\beta\eta$-well-typed in $\beta\eta(\sigma'\Gamma)$, and thus

$$\overline{\Gamma'} \vdash Q' : \forall\overline{z}_n : \overline{A}_n.\forall\overline{x}_p : \overline{C}_p.\langle x_1,\ldots,x_p\rangle \in N \to Px_1\ldots x_p.$$

holds. This judgment is equivalent to

$$\overline{\Gamma'} \vdash Q' : \forall\overline{z}_n : \overline{A}_n.\forall\overline{x}_p : \overline{C}_p.\langle x_1,\ldots,x_p\rangle \in N \to \langle x_1\ldots x_p\rangle \in \{\langle x_1,\ldots,x_p\rangle \mid Px_1\ldots x_p\}$$

which is equivalent to

$$\overline{\Gamma'} \vdash Q' : \forall\overline{z}_n : \overline{A}_n.N \subseteq \{\langle x_1,\ldots,x_p\rangle \mid Px_1\ldots x_p\}.$$

Hence Q' must have the form $\lambda\overline{z}_n : \overline{A}_n.Q''$ where Q'' is in normal form and the following also holds:

$$\overline{\Gamma'}, \Phi \vdash Q'' : N \subseteq \{\langle x_1,\ldots,x_p\rangle \mid Px_1\ldots x_p\}. \tag{5}$$

Using (3) and (5), we can take Q in (4) to be

$$\lambda C : Prop.\lambda f : (\{\langle x_1,\ldots,x_p\rangle \mid Px_1\ldots x_p\} \subseteq N) \to (N \subseteq \{\langle x_1,\ldots,x_p\rangle \mid Px_1\ldots x_p)\} \to C.fP'Q''$$

and we have our result. ■

Theorem 18. *Let Φ and Φ' be contexts of the form $z_1 : A_1,\ldots,z_n : A_n$ and $x_1 : C_1,\ldots,x_j : C_j$ respectively, where $n, j \geq 0$. Let Γ be a normal valid search context of the form*

$$\Gamma', (\Phi, z, B), ((\Phi,\Phi'), h, \langle x_1,\ldots,x_j, f_1x_1\ldots x_p,\ldots,f_rx_1\ldots x_p\rangle \in zz_1\ldots z_n \to P'x_1\ldots x_p)$$

for some $p > j$ and $r > 0$ such that Γ' does not contain any existential or constraint triples, B is a set type of the form $\forall x_1 : C_1\ldots\forall x_j : C_j.C_{j+1} \to \cdots \to C_{j+r} \to Prop$, the terms in Φ' contain no free occurrences of z, and the following judgments hold

$$\begin{array}{l}\overline{\Gamma}, \Phi \vdash P' : \forall x_1 : C_1\ldots\forall x_j : C_j.\forall x_{j+1} : D_{j+1}\ldots\forall x_p : D_p.Prop \\ \overline{\Gamma}, \Phi \vdash f_i : \forall x_1 : C_1\ldots\forall x_j : C_j.\forall x_{j+1} : D_{j+1}\ldots\forall x_p : D_p.C_{j+i} \quad \textit{for } i = 1,\ldots,r.\end{array}$$

$$\{\langle x_1,\ldots,x_j,w_1,\ldots,w_r\rangle \mid \forall x_{j+1}\!:\!D_{j+1}\ldots\forall x_p\!:\!D_p.$$
$$w_1 =_{C_{j+1}} f_1x_1\ldots x_p \to \cdots \to w_r =_{C_{j+r}} f_rx_1\ldots x_p \to P'x_1\ldots x_p\}$$

is a maximal solution for $zz_1\ldots z_n$ *in* Γ.

Proof. Let σ and τ be the substitutions containing the single tuples

$$\begin{array}{l}
\langle z,\langle\rangle,\lambda\overline{z}_n\!:\!\overline{A}_n.\{\langle x_1,\ldots,x_j,w_1,\ldots,w_r\rangle \mid \\
\quad \forall x_{j+1}\!:\!D_{j+1}\ldots\forall x_p\!:\!D_p.w_1 =_{C_{j+1}} f_1x_1\ldots x_p \to \cdots \to w_r =_{C_{j+r}} f_rx_1\ldots x_p \to P'x_1\ldots x_p\}\rangle \\
\langle h,\langle\rangle,\lambda\overline{z}_n\!:\!\overline{A}_n.\lambda\overline{x}_j\!:\!\overline{C}_j.\lambda f\!:\!(\forall x_{j+1}\!:\!D_{j+1}\ldots\forall x_p\!:\!D_p. \\
\quad f_1x_1\ldots x_p =_{C_{j+1}} f_1x_1\ldots x_p \to \cdots \to f_rx_1\ldots x_p =_{C_{j+r}} f_rx_1\ldots x_p \to P'x_1\ldots x_p). \\
\quad fx_{j+1}\ldots x_p(\lambda P\!:\!C_{j+1} \to Prop.\lambda x\!:\!P(f_1x_1\ldots x_p).x)\ldots(\lambda P\!:\!C_{j+r} \to Prop.\lambda x\!:\!P(f_rx_1\ldots x_p).x)\rangle
\end{array}$$

respectively. As in the proof of Theorem 17, we can show that τ is a solution to the normal form of $\sigma\Gamma$, and that the solution for $zz_1\ldots z_n$ is maximal. ■

Theorem 19. *Let* Φ *and* Φ' *be contexts of the form* $z_1 : A_1,\ldots,z_n : A_n$ *and* $x_1 : C_1,\ldots,x_j : C_j$ *respectively, where* $n, j \geq 0$. *Let* Γ *be a normal valid search context of the form*

$$\Gamma',(\Phi,z,B),((\Phi,\Phi'),h,\langle x_1,\ldots,x_j,M_1,\ldots,M_r\rangle \in zz_1\ldots z_n \to Q)$$

for some $r > 0$ *such that* Γ' *does not contain any existential or constraint triples,* B *is a set type of the form* $\forall x_1\!:\!C_1\ldots\forall x_j\!:\!C_j.C_{j+1} \to \cdots \to C_{j+r} \to Prop$, *the judgment* $\overline{\Gamma'},\Phi \vdash Q : Prop$ *holds, and the judgments* $\overline{\Gamma'},\Phi,\Phi' \vdash M_i : C_{j+i}$ *hold for* $i = 1,\ldots,r$, *and the terms in* Φ' *contain no free occurrences of* z. *Then* $\{\langle x_1,\ldots,x_j,w_1,\ldots,w_r\rangle \mid w_1 =_{C_{j+1}} M_1 \to \cdots \to w_r =_{C_{j+r}} M_r \to Q\}$ *is a maximal solution for* $zz_1\ldots z_n$ *in* Γ.

Proof. Let σ and τ be the substitutions containing the single tuples

$$\begin{array}{l}
\langle z,\langle\rangle,\lambda\overline{z}_n\!:\!\overline{A}_n.\{\langle x_1,\ldots,x_j,w_1,\ldots,w_r\rangle \mid w_1 =_{C_{j+1}} M_1 \to \cdots \to w_r =_{C_{j+r}} M_r \to Q\}\rangle \\
\langle h,\langle\rangle,\lambda\overline{z}_n\!:\!\overline{A}_n.\lambda\overline{x}_j\!:\!\overline{C}_j.\lambda f\!:\!(M_1 =_{C_{j+1}} M_1 \to \cdots \to M_r =_{C_{j+r}} M_r \to Q). \\
\quad f(\lambda P\!:\!C_{j+1} \to Prop.\lambda x\!:\!PM_1.x)\ldots(\lambda P\!:\!C_{j+r} \to Prop.\lambda x\!:\!PM_r.x)\rangle
\end{array}$$

respectively. As in the previous theorems, we can show that τ is a solution to the normal form of $\sigma\Gamma$, and that the solution for $zz_1\ldots z_n$ is maximal. ■

Theorem 20. *Let* Φ *and* Φ' *be contexts of the form* $z_1 : A_1,\ldots,z_n : A_n$ *and* $x_1 : C_1,\ldots,x_j : C_j$ *respectively, where* $n, j \geq 0$. *Let* Γ *be a normal valid search context of the form*

$$\Gamma',(\Phi,z,B),((\Phi,\Phi'),h,\neg(\langle x_1,\ldots,x_j,M_1,\ldots,M_r\rangle \in zz_1\ldots z_n)$$

for some $r > 0$ *such that* Γ' *does not contain any existential or constraint triples,* B *is a set type of the form* $\forall x_1 : C_1\ldots\forall x_j : C_j.C_{j+1} \to \cdots \to C_{j+r} \to Prop$, *the judgments* $\overline{\Gamma'},\Phi,\Phi' \vdash M_i : C_{j+i}$ *hold for* $i = 1,\ldots,r$, *and the terms in* Φ' *contain no free occurrences of* z. *Then* $\{\langle x_1,\ldots,x_j,w_1,\ldots,w_r\rangle \mid \neg(w_1 =_{C_{j+1}} M_1 \to \cdots \to w_r =_{C_{j+r}} M_r)\}$ *is a maximal solution for* $zz_1\ldots z_n$ *in* Γ.

Proof. This theorem is an instance of Theorem 19 with $\perp$ as an instance of Q. ■

valid search context of the form

$$\Gamma', (\Phi, z, A), (\Phi, h, \langle N_1, \ldots, N_p\rangle \in zz_1 \ldots z_n)$$

for some $p > 0$ such that Γ' does not contain any existential or constraint triples, A is a set type of the form $\forall x_1 : C_1 \ldots \forall x_p : C_p.Prop$, and for $i = 1, \ldots, p$, the judgments $\overline{\Gamma'}, \Phi \vdash N_i : [N_1/x_1, \ldots, N_{i-1}/x_{i-1}]C_i$ hold. Then $\{\langle x_1, \ldots, x_p\rangle \mid \top\}$ is a maximal solution for $zz_1 \ldots z_n$ in Γ.

Proof. Let σ and τ be the substitutions containing the single tuples

$$\langle z, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n.\{\langle x_1, \ldots, x_p\rangle \mid \top\}\rangle \text{ and } \langle h, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n.\lambda C : Prop.\lambda x : C.x\rangle,$$

respectively. As in the previous theorems, we can show that τ is a solution to the normal form of $\sigma\Gamma$. To show that the universal set $\{\langle x_1, \ldots, x_p\rangle \mid \top\}$ is maximal, we simply show that for any set N, $N \subseteq \{\langle x_1, \ldots, x_p\rangle \mid \top\}$. The following is the judgment stating this fact.

$$\begin{aligned} \overline{\Gamma'}, \Phi \quad \vdash \quad & \lambda x_1 : C_1 \ldots \lambda x_p : C_p.\lambda x' : N x_1 \ldots x_p.\lambda C : Prop.\lambda x : C.x : \\ & \forall x_1 : C_1 \ldots \forall x_p : C_p.N x_1 \ldots x_p \to \forall C : Prop.C \to C \end{aligned}$$

■

Theorem 22. *Let Φ be a context of the form $z_1 : A_1, \ldots, z_n : A_n$ where $n \geq 0$. Let Γ be a normal valid search context of the form*

$$\Gamma', (\Phi, z, A), (\Phi, h, P \wedge Q)$$

such that Γ' does not contain any existential or constraint triples, A is a set type of the form $\forall x_1 : C_1 \ldots \forall x_p : C_p.Prop$ for some $p > 0$, and $\overline{\Gamma'}, \exists z : \forall \overline{z}_n : \overline{A}_n.A, \Phi \vdash P \wedge Q : Prop$ holds. Let C' and h' be variables that do not occur in Γ, let Φ' be the context $C' : Prop, h' : P \to Q \to C'$, and let D_1 and D_2 be maximal solutions for A in

$$\Gamma', (\Phi, z, A), ((\Phi, \Phi'), h, P) \qquad \textit{and} \qquad \Gamma', (\Phi, z, A), ((\Phi, \Phi'), h, Q),$$

respectively. Then $D_1 \cap D_2$ is a maximal solution for A in Γ.

Proof. Let Γ_1 and Γ_2 be the contexts $\Gamma', (\Phi, z, A), ((\Phi, \Phi'), h, P)$ and $\Gamma', (\Phi, z, A), ((\Phi, \Phi'), h, Q)$, respectively. Let σ_1, σ_2, and σ be the substitutions containing the single tuples

$$\langle z, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n.D_1\rangle, \qquad \langle z, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n.D_2\rangle, \qquad \langle z, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n.D_1 \cap D_2\rangle,$$

respectively. Note that $D_1 \cap D_2$ is an abbreviation for

$$\lambda x_1 : C_1 \ldots \lambda x_p : C_p.\forall C' : Prop.(D_1 x_1 \ldots x_p \to D_2 x_1 \ldots x_p \to C') \to C'.$$

Because D_1 and D_2 are maximal solutions for A in Γ_1 and Γ_2, respectively, we know that there are terms M_1 and M_2 and substitutions τ_1 and τ_2 defined as follows,

$$\begin{aligned} &\langle h, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n.\lambda C' : Prop.\lambda h' : P \to Q \to C'.M_1\rangle \\ &\langle h, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n.\lambda C' : Prop.\lambda h' : P \to Q \to C'.M_2\rangle \end{aligned}$$

solution to the normal form of $\sigma_2\Gamma_2$. Using these substitutions, it is straightforward to show that

$$\langle h, \langle\rangle, \lambda\overline{z}_n : \overline{A}_n . \lambda C' : Prop . \lambda h' : P \to Q \to C' . h' M_1 M_2 \rangle .$$

is a solution to the normal form of $\sigma\Gamma$.

The proof that $D_1 \cap D_2$ is maximal follows the same reasoning as the corresponding proof in Bledsoe [3]. Since no extensions are needed to adapt this proof to our setting beyond what already appears in the proof of Theorem 17, we omit the details. ■

4 A Complete Search Procedure

To incorporate the full expressiveness of CC, we extend CC$^+$ to Meta as defined in [10]. This inference system includes all the rules for CC$^+$ plus the following additional rule where *Extern* is a new sort.

$$\frac{\vdash \Gamma \ \mathsf{context}}{\Gamma \vdash \mathit{Type} : \mathit{Extern}} \ (\text{TYPE-EXTERN})$$

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*. In this section, validity of standard and search contexts will be with respect to Meta.

The SETVAR, INTRO, and BACKCHAIN operations are sufficient for proving the examples given in Sect. 1 as well as most of the examples in Bledsoe [3] and they are also the ones implemented in our λProlog implementation. We add the SPLIT, PROD, and POLY operations below to obtain the SetVar$^+$ procedure that is complete for the full CC. As stated earlier, they add complications for directing search. For example, once POLY becomes applicable, it is possible to apply it infinitely many times.

With the addition of the three new operations, we no longer need SETVAR. The procedure is complete without it. We leave it in because even in the context of a complete procedure, it is useful for directing search towards finding certain substitution instances more quickly. The other operations are useful for the cases when SETVAR isn't enough. Since it is not needed, SETVAR does not appear in the proof of completeness of SetVar$^+$. It's soundness was already established in the previous section.

SPLIT operation. Let Γ be a valid search context and $((z_1 : A_1, \ldots, z_n : A_n), z, xM_1 \ldots M_m)$ a candidate triple in Γ, where $m, n \geq 0$, and $\overline{\Gamma} \vdash xM_1 \ldots M_m : s$ holds where s is *Prop* or *Type*. If there is a universal declaration $w : Q$ such that either w is one of $z_1, \ldots, z_n$ or $w : Q$ occurs to the left of $((z_1 : A_1, \ldots, z_n : A_n), z, xM_1 \ldots M_m)$ in Γ, the judgment $\overline{\Gamma}, z_1 : A_1, \ldots, z_n : A_n \vdash Q : s$ holds, Q has the form $\forall y_1 : Q_1 \ldots \forall y_q : Q_q . yN_1 \ldots N_p$ $(p, q \geq 0)$, and y is any existential variable in Γ, then let $h_1, \ldots, h_q$ be variables that do not occur in Γ. Let Φ be the context $z_1 : A_1, \ldots, z_n : A_n$. Let Δ_0 be the context

$$\begin{array}{l}(\Phi, h_1, Q_1), \\ (\Phi, h_2, [h_1 z_1 \ldots z_n / y_1] Q_2), \\ \quad \vdots \\ (\Phi, h_q, [h_1 z_1 \ldots z_n / y_1, \ldots, h_{q-1} z_1 \ldots z_n / y_{q-1}] Q_q).\end{array}$$

Choose a j such that $j > 0$. For $i = 1, \ldots, j$, let s_i be either *Prop* or *Type*. Let $H_1, \ldots, H_j$, $K_1, \ldots, K_j$, $h_{q+1}, \ldots, h_{q+j}$ be variables that do not occur in Γ. For $i = 1, \ldots, j$, let Δ_i be the

$$\begin{array}{l}(\Phi, H_i, s_i),\\(\Phi, K_i, H_i z_1 \dots z_n \to s),\\(\Phi, L, \forall u\!:\!H_i z_1 \dots z_n . K_i z_1 \dots z_n u)\\(\Phi, h_{q+i}, H_i z_1 \dots z_n)\end{array}$$

where if $i = 1$, L is the term $[h_1 z_1 \dots z_n / y_1, \dots, h_q z_1 \dots z_n / y_q] y N_1 \dots N_p$ and if $i > 1$, L is the term $K_{i-1} z_1 \dots z_n (h_{q+i-1} z_1 \dots z_n)$. Let Δ' be the context

$$(\Phi, K_j z_1 \dots z_n (h_{q+j} z_1 \dots z_n), x M_1 \dots M_m).$$

Let Δ be the context $\Delta_0, \Delta_1, \dots, \Delta_j, \Delta'$. Let σ be the substitution

$$\{\langle z, \Delta, \lambda \overline{z}_n\!:\!\overline{A}_n . w (h_1 z_1 \dots z_n) \dots (h_{q+j} z_1 \dots z_n)\rangle\}.$$

PROD operation. Let Γ be a valid search context and $((z_1\!:\!A_1, \dots, z_n\!:\!A_n), z, s')$ a candidate triple in Γ, where $n \geq 0$ and s' is *Type* or *Extern*. Let s be the sort such that $\Gamma \vdash s : s'$. Let σ be the substitution $\{\langle z, \langle\rangle, \lambda \overline{z}_n\!:\!\overline{A}_n . s\rangle\}$.

POLY operation. Let Γ be a valid search context and $((z_1\!:\!A_1, \dots, z_n\!:\!A_n), z, s')$ a candidate triple in Γ, where $n \geq 0$, and s' is any sort. Let s be *Prop* or *Type* and let Φ be the context $z_1\!:\!A_1, \dots, z_n\!:\!A_n$. Let h and k be variables that do not occur in Γ. Let Δ be the context

$$(\Phi, h, s), (\Phi, k, h z_1 \dots z_n \to s').$$

Let σ be the substitution $\{\langle z, \Delta, \lambda \overline{z}_n\!:\!\overline{A}_n . \forall u\!:\!h z_1 \dots z_n . k z_1 \dots z_n u\rangle\}$.

The SPLIT operation can be viewed as an extension of BACKCHAIN. If j were allowed to be 0 in this operation, the operation essentially reduces to BACKCHAIN. We illustrate its use by returning to the example from Sect. 3.1 for which INTRO and BACKCHAIN were not sufficient. The following intermediate context appeared in the example as context (12).

$$\Gamma, ((x\!:\!Nat), A_0, Prop), ((x\!:\!Nat, h\!:\!A_0 x), M'_1, Px), ((x\!:\!Nat, h\!:\!A_0 x), M'_2, Qx) \tag{1}$$

Consider the second existential triple as a candidate triple for the SPLIT operation. The universal declaration used in this operation will be $h\!:\!A_0 x$ from the local context. We choose j to be 1 and s_1 to be *Prop*. The context Δ_0 of this operation is empty in this case and Δ_1 is as follows

$$\begin{array}{rcl}\Delta_1 & := & ((x\!:\!Nat, h\!:\!A_0 x), H_1, Prop),\\ & & ((x\!:\!Nat, h\!:\!A_0 x), K_1, H_1 x h \to Prop),\\ & & ((x\!:\!Nat, h\!:\!A_0 x), A_0 x, \forall u\!:\!H_1 x h . K_1 x h u),\\ & & ((x\!:\!Nat, h\!:\!A_0 x), h_1, H_1 x h)\end{array}$$

where H_1, K_1, h_1 are new variables. Δ is obtained by adding $((x\!:\!Nat, h\!:\!A_0 x), K_1 x h (h_1 x h), Px)$ to the end of Δ_1. The substitution σ generated by this operation is

$$\sigma := \{\langle M'_1, \Delta, \lambda x\!:\!Nat . \lambda h\!:\!A_0 x . h (h_1 x h)\rangle\}.$$

Applying σ to (1), we get

$$\begin{array}{l}\Gamma, ((x\!:\!Nat), A_0, Prop), ((x\!:\!Nat, h\!:\!A_0 x), H_1, Prop),\\ \quad ((x\!:\!Nat, h\!:\!A_0 x), K_1, H_1 x h \to Prop), ((x\!:\!Nat, h\!:\!A_0 x), A_0 x, \forall u\!:\!H_1 x h . K_1 x h u),\\ \quad ((x\!:\!Nat, h\!:\!A_0 x), h_1, H_1 x h), ((x\!:\!Nat, h\!:\!A_0 x), K_1 x h (h_1 x h), Px),\\ \quad ((x\!:\!Nat, h\!:\!A_0 x), M'_2, Qx).\end{array}$$

operation must be used to obtain an instantiation for A_0 that can lead to a context in which this constraint is satisfied. We illustrate by going back to the context (1), and considering the first existential triple as the candidate triple. Let s of POLY be *Prop*. We obtain the following context and substitution

$$\begin{array}{l}\Delta := ((x\!:\!Nat), h', Prop), ((x\!:\!Nat), k, (h'x \to Prop)) \\ \sigma := \{\langle A_0, \Delta, \lambda x\!:\!Nat.\forall u\!:\!h'x.kxu\rangle\}\end{array}$$

where h' and k are new variables. Applying σ to (1), we get

$$\begin{array}{l}\Gamma,((x\!:\!Nat), h', Prop), ((x\!:\!Nat), k, (h'x \to Prop)), \\ \quad ((x\!:\!Nat, h\!:\!(\forall u\!:\!h'x.kxu)), M_1', Px), ((x\!:\!Nat, h\!:\!(\forall u\!:\!h'x.kxu)), M_2', Qx)\end{array}$$

Note here that A_0x has been replaced by the non-atomic type $\forall u\!:\!h'x.kxu$.

To prove correctness of SetVar$^+$, we prove soundness by extending Theorem 16 for SetVar to the new operations, and we prove completeness relative to Dowek's procedure.

Theorem 23. *(Soundness of SetVar$^+$) Let Γ be a normal valid Meta context without existential variables or constraints such that the types of universal variables in declarations are Prop or Type but not Extern. Let A be a normal well-typed term in Γ. Let Γ' be the search context $\Gamma, (\langle\rangle, z, A)$. If there exists a derivation of Γ', then there exists a term M such that $\Gamma \vdash M : A$ holds in CC.*

Proof. The properties in Sect. 3.2 about search contexts in CC$^+$ also hold for search contexts of Meta. We only need to extend Lemmas 12 and 13 with cases for SPLIT, PROD, and POLY. Since these cases follow similarly to the cases already shown, we omit the details. Once these lemmas are extended, Lemma 14 and 15 and Theorem 16 follow directly for the extended search procedure. ■

To prove completeness we introduce Dowek's procedure, which we call $\mathcal{P}$. $\mathcal{P}$ operates directly on Meta contexts. These contexts are restricted so that the types of universal variables in declarations are *Prop* or *Type* but not *Extern*. We define a *candidate declaration* in a standard Meta context Γ to be an existential declaration of the form $\exists z\!:\!(\forall z_1\!:\!A_1 \ldots \forall z_n\!:\!A_n.xM_1 \ldots M_p)$ where $n, p \geq 0$ and x is universal in $\Gamma, z_1 : A_1, \ldots, z_n : A_n$. Like SetVar$^+$, at each step, a search operation is applied resulting in a substitution. Note that since only variables in existential declarations can have type *Extern*, if the procedure leads to a success context, all such variables will be instantiated eliminating all occurrences of *Extern* and resulting in a valid CC context.

The procedure is defined by the three search operations given below. The first combines INTRO, BACKCHAIN, and SPLIT, while the other two correspond directly to PROD and POLY.

1. Let Γ be a valid Meta context and $\exists z\!:\!P$ a candidate declaration in Γ, where P has the form $\forall z_1 : A_1 \ldots \forall z_n : A_n.xM_1 \ldots M_m$ $(m, n \geq 0)$ and $\Gamma \vdash P : s$ holds where s is any sort (including *Extern*). This operation applies if there is a universal declaration $w : Q$ such that either w is one of $z_1, \ldots, z_n$ or $w\!:\!Q$ occurs to the left of this candidate declaration in Γ, Q has the form $\forall y_1 : Q_1 \ldots \forall y_q : Q_q.yN_1 \ldots N_p$ $(p, q \geq 0)$, and $\Gamma \vdash Q : s$ holds. Let $h_1, \ldots, h_q$ be variables that do not occur in Γ. Let Δ_0 be the context

$$\begin{array}{l}\exists h_1\!:\!\forall\overline{z}_n\!:\!\overline{A}_n.Q_1, \\ \exists h_2\!:\!\forall\overline{z}_n\!:\!\overline{A}_n.[h_1z_1 \ldots z_n/y_1]Q_2, \\ \quad\vdots \\ \exists h_q\!:\!\forall\overline{z}_n\!:\!\overline{A}_n.[h_1z_1 \ldots z_n/y_1, \ldots, h_{q-1}z_1 \ldots z_n/y_{q-1}]Q_q.\end{array}$$

$K_1, \ldots, K_j$, $h_{q+1}, \ldots, h_{q+j}$ be variables that do not occur in Γ. For $i = 1, \ldots, j$, let Δ_i be the following context

$$\begin{array}{l} \exists H_i : \forall \overline{z}_n : \overline{A}_n . s_i, \\ \exists K_i : \forall \overline{z}_n : \overline{A}_n . H_i z_1 \ldots z_n \rightarrow s, \\ \forall \overline{z}_n : \overline{A}_n . L = \forall \overline{z}_n : \overline{A}_n . \forall u : H_i z_1 \ldots z_n . K_i z_1 \ldots z_n u \\ \exists h_{q+i} : \forall \overline{z}_n : \overline{A}_n . H_i z_1 \ldots z_n \end{array}$$

where if $i = 1$, L is the term $[h_1 z_1 \ldots z_n / y_1, \ldots, h_q z_1 \ldots z_n / y_q] y N_1 \ldots N_p$ and if $i > 1$, L is the term $K_{i-1} z_1 \ldots z_n (h_{q+i-1} z_1 \ldots z_n)$. If $r = 0$, let Δ' be the context

$$\forall \overline{z}_n : \overline{A}_n . [h_1 z_1 \ldots z_n / y_1, \ldots, h_q z_1 \ldots z_n / y_q] y N_1 \ldots N_n = \forall \overline{z}_n : \overline{A}_n . x M_1 \ldots M_m .$$

Otherwise, let Δ' be the context

$$(\forall \overline{z}_n : \overline{A}_n . K_j z_1 \ldots z_n (h_{q+j} z_1 \ldots z_n) = \forall \overline{z}_n : \overline{A}_n . x M_1 \ldots M_m).$$

Let Δ be the context $\Delta_0, \Delta_1, \ldots, \Delta_j, \Delta'$. Let σ be the substitution:

$$\{\langle z, \Delta, \lambda \overline{z}_n : \overline{A}_n . w (h_1 z_1 \ldots z_n) \ldots (h_{q+j} z_1 \ldots z_n) \rangle\}.$$

2. Let Γ be a valid search context and $\exists z : P$ a candidate declaration in Γ, where P has the form $\forall z_1 : A_1 \ldots \forall z_n : A_n . s'$ ($n \geq 0$) and s' is *Type* or *Extern*. Let s be the sort such that $\Gamma \vdash s : s'$. Let σ be the substitution $\{\langle z, \langle\rangle, \lambda \overline{z}_n : \overline{A}_n . s \rangle\}$.

3. Let Γ be a valid search context and $\exists z : P$ a candidate declaration in Γ, where P has the form $\forall z_1 : A_1 \ldots \forall z_n : A_n . s'$ ($n \geq 0$) and s' is any sort. Let s be *Prop* or *Type* and let Δ be the context
$$\exists h : \forall \overline{z}_n : \overline{A}_n . s, \exists k : \forall \overline{z}_n : \overline{A}_n . h z_1 \ldots z_n \rightarrow s'.$$
Let σ be the substitution $\{\langle z, \Delta, \lambda \overline{z}_n : \overline{A}_n . \forall u : h z_1 \ldots z_n . k z_1 \ldots z_n u \rangle\}$.

To prove completeness of SetVar$^+$, in the following lemma we show that every operation that can be performed on a standard context in $\mathcal{P}$ has a corresponding operation or set of operations on search contexts in SetVar$^+$. The lemma is stated using standard contexts. For a standard context Γ, when applying operations of SetVar$^+$, Γ is viewed as the context such that every existential declaration of the form $\exists z : A$ is replaced by $(\langle\rangle, z, A)$ and every constraint $P = Q$ is replaced by $(\langle\rangle, P, Q)$.

Lemma 24. *Let Γ be a valid Meta context such that the types of universal variables in declarations are Prop or Type but not Extern. If σ is the result of applying a search operation in $\mathcal{P}$, then it is either the case that subsequent operations to $\sigma\Gamma$ always lead to a failure context or there is a series of operations in SetVar$^+$ with substitutions $\tau_1, \ldots, \tau_n$ such that the normal forms of $\sigma\Gamma$ and $\overline{(\tau_1 \circ \cdots \circ \tau_n)\Gamma}$ are the same context.*

Proof. Let $\langle\langle\rangle, z, Q\rangle$ be the candidate declaration to which the operation in $\mathcal{P}$ is applied. Γ has the form $\Gamma', \langle\langle\rangle, z, Q\rangle, \Gamma''$. For the case when the operation applied is the first operation of $\mathcal{P}$, Q has the form
$$\forall z_1 : A_1 \ldots \forall z_n : A_n . x M_1 \ldots M_m .$$

can first apply INTRO n times with substitutions $\tau_1, \ldots, \tau_n$ where for $i = 1, \ldots, n$, τ_i is

$$\{\langle z'_{i-1}, ((z_1 : A_1, \ldots, z_i : A_i), z'_i, \forall z_{i+1} : A_{i+1} \ldots \forall z_n : A_n . x M_1 \ldots M_m)\rangle\}$$

where z is z'_0 and $z'_1, \ldots, z'_n$ are new variables. We obtain the context

$$\Gamma', ((z_1 : A_1, \ldots, z_n : A_n), z'_n, x M_1 \ldots M_m), \Gamma''.$$

We first consider the case when j of the first operation of $\mathcal{P}$ is 0. If x is w or an existential variable, then we apply BACKCHAIN in SetVar$^+$ to obtain substitution σ' where the context Δ in the tuple in σ is the same as $\overline{\Delta}$ in σ'. In particular, if σ' is the substitution $\{\langle z'_n, \Delta, M\rangle\}$ for some term M, then σ is the substitution $\{\langle z, \overline{\Delta}, M\rangle\}$. Note that $\overline{\sigma'}$ is $\overline{\{\langle z'_n, \overline{\Delta}, M\rangle\}}$ and thus $\overline{\sigma'}$ differs from σ only in the name of the variable it binds. We show that $\overline{(\tau_1 \circ \cdots \circ \tau_n \circ \sigma')\Gamma}$ is the same context as $\sigma\Gamma$. By Lemma 9, $(\tau_1 \circ \cdots \circ \tau_n \circ \sigma')\Gamma$ is $\sigma'\tau_n \cdots \tau_1\Gamma$, and so by Lemma 5, $\overline{\sigma'\tau_n \cdots \tau_1\Gamma}$ is $\overline{\sigma'}(\overline{\tau_n \cdots \tau_1\Gamma})$. By a simple induction on n, we can show that for $i = 1, \ldots, n$, the context $\overline{\tau_i \cdots \tau_1\Gamma}$ is $\Gamma', \langle\langle\rangle, z'_i, Q\rangle, [z'_i/z]\Gamma''$. Thus, $\overline{\sigma'}(\overline{\tau_n \cdots \tau_1\Gamma})$ is $\Gamma', \overline{\Delta}, [M/z'_n]([z'_n/z]\Gamma'')$. Since $[M/z'_n]([z'_n/z]\Gamma'')$ is just $[M/z]\Gamma''$, it is easy to see that this context is also $\sigma\Gamma$ and thus $\overline{(\tau_1 \circ \cdots \circ \tau_n \circ \sigma')\Gamma}$ is the same context as $\sigma\Gamma$.

For the case when $j = 0$ and x is universal and different from w (which is allowed in $\mathcal{P}$), it is easy to see that the resulting context leads to a failure context; once the existential variables that remain in the constraint that gets added by applying the substitution are fully instantiated, this constraint will relate two terms that are not $\beta\eta$-convertible.

For the case when $j > 0$, the first operation of $\mathcal{P}$ corresponds to a series of n applications of INTRO, followed by the SPLIT operation in SetVar$^+$. Similar reasoning can be applied to show that $\sigma\Gamma$ is $\overline{(\tau_1 \circ \cdots \circ \tau_n)\Gamma}$.

Similarly, the cases for the second and third operations of $\mathcal{P}$ correspond to a series of applications of INTRO followed by an application of PROD or POLY, respectively. ■

Theorem 25. *(Completeness) Let Γ be a valid Meta context without existential variables or constraints such that the types of universal variables in declarations are Prop or Type but not Extern. Let A be a normal well-typed term in Γ. If there exists a derivation of $\Gamma, \exists z : A$ in $\mathcal{P}$, then there is a derivation of $\Gamma, (\langle\rangle, z, A)$ in SetVar$^+$.*

Proof. We prove the following stronger statement. Let Γ be an arbitrary normal valid search context such that the types of universal variables in declarations are *Prop* or *Type* but not *Extern*. If $\overline{\Gamma}$ has a derivation in $\mathcal{P}$, then Γ has a derivation in SetVar$^+$. The proof is by induction on the length of a derivation in $\mathcal{P}$. The desired theorem follows directly. ■

5 Conclusion

We have shown how to adapt Bledsoe's method for generating maximal solutions for set variables to the Calculus of Constructions and proved its correctness. In addition, we have discussed the operation of the procedure on various sublanguages. The procedure presented here has been implemented as a set of tactics within an interactive tactic-style theorem prover. These tactics can be combined to automate the search procedure for CC so that it works efficiently on the class of theorems involving existential quantification over sets. It can also be used as a tactic in Coq to provide some automation for this class of theorems.

rules for conjunction were adapted fairly directly, while the combining rules for disjunction were handled in a distributed manner. The remaining rules in Bledsoe [3] 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 clauses to the SETVAR operation.

We have shown how one procedure designed for a higher-order logic can be carried over to the type theory setting. There are many other interesting procedures worth investigation. Bledsoe and Feng give a more general set of rules for maximal solutions in [4]. This procedure, however, relies heavily on resolution techniques which may be difficult to adapt to our setting. Another procedure for automating the instantiation of set variables is the $\mathcal{Z}$-match inference rule in [1], which should be possible to adapt to our setting fairly directly. In addition, many other theorem proving techniques in a variety of domains 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 setting.

Acknowledgements

The author would like to thank the anonymous reviewers for numerous helpful suggestions.

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] H. Barendregt. Introduction to generalized type systems. *Journal of Functional Programming*, 1(2):124–154, April 1991.

[3] W. W. Bledsoe. A maximal method for set variables in automatic theorem proving. *Machine Intelligence*, 9:53–100, 1979.

[4] W. W. Bledsoe and G. Feng. SET-VAR. *Journal of Automated Reasoning*, 11(3):293–314, 1993.

[5] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[6] R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, 1986.

[7] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, February/March 1988.

[8] 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.

Université Paris VII, Dec. 1991.

[10] G. Dowek. A complete proof synthesis method for the cube of type systems. *Journal of Logic and Computation*, 3(3):287–315, 1993.

[11] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. In *Tenth Annual Symposium on Logic in Computer Science*, pages 366–374, 1995.

[12] 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.

[13] A. Felty. Implementing tactics and tacticals in a higher-order logic programming language. *Journal of Automated Reasoning*, 11(1):43–81, Aug. 1993.

[14] A. Felty. Proof search with set variable instantiation in the calculus of constructions. In *Thirteenth International Conference on Automated Deduction*, pages 658–672. Springer-Verlag Lecture Notes in Computer Science, July 1996.

[15] M. J. C. Gordon and T. F. Melham. *Introduction to HOL—A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[16] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, Jan. 1993.

[17] 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.

[18] G. Huet. A uniform approach to type theory. In G. Huet, editor, *Logical Foundations of Functional Programming*. Addison Wesley, 1990.

[19] L. Magnusson. *The Implementation of ALF: A Proof Editor Based on Martin-Löf's Monomorphic Type Theory with Explicit Substitution*. PhD thesis, Chalmers University of Technology/Göteborg University, Jan. 1995.

[20] P. Martin-Löf. *Intuitionistic Type Theory*. Studies in Proof Theory Lecture Notes. BIBLIOPOLIS, Napoli, 1984.

[21] 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.

[22] C. Muñoz. *A Calculus of Substitutions for Incomplete-Proof Representation in Type Theory*. PhD thesis, Université Paris 7, INRIA Research Report RR-3309 (English version), 1997.

[23] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.