Electronic Notes in Theoretical Computer Science 70 No. 2 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume70.html 22 pages

Towards Proof Planning for $\mathcal{M}_\omega^+$ ⋆

Carsten Schürmann [1]

Yale University, New Haven (CT), USA

Serge Autexier [2]

German Research Center for Artificial Intelligence (DFKI)
Saarbrücken, Germany

Abstract

This paper describes the proof planning system $\mathcal{P}_\omega^+$ for the meta theorem prover for LF implemented in Twelf. The main contributions include a formal system that approximates the flow of information between assumptions and goals within a meta proof, a set of inference rules to reason about those approximations, and a soundness proof that guarantees that the proof planner does not reject promising proof states. Proof planning in $\mathcal{P}_\omega^+$ is decidable.

1 Introduction

The difficulty and sheer complexity of automated deduction tasks are so daunting that implementers of many interactive theorem proving systems such as Isabelle [Pau94], Coq [DFH+93], Lego [LP92], PVS [ORS92], and INKA [Hut96] prove lemmas and theorems with heuristics based techniques. Tactics [GW79], for example, heuristically explore the search space until no more progress can be made, and proof planning techniques [Bun88] heuristically plan a path through the search space of states to guide proof search.

The degree and the success of automation that one can expect from a theorem proving system seems directly related to the way a problem is represented. Many fully automatic theorem provers employ logic, either first-order and less often higher-order logics with or without equality as the representation language. As a result the resulting theorem provers are typically highly specialized for one particular class of problems.

⋆ This work was supported in part by the National Science Foundation NSF under grants CCR-0133502 and INT-9909952, and by the German Academic Exchange Service DAAD.

[1] Email: carsten@cs.yale.edu

[2] Email: autexier@dfki.de

Others such as the Twelf system [PS99] use logical framework technology [Pfe99] for representing the domains in which theorem proving is to be conducted. Those domains include various deductive systems, such as logics and type systems. In addition, recent work on theorem proving about encodings that are represented in LF using higher-order representation techniques have led to the meta logic $\mathcal{M}_\omega^+$ [Sch00] and a prototype implementation as part of the Twelf implementation.

The quintessential difference between theorem proving in first-order logics where the subjects of reasoning are encoded using relations, and theorem proving in logical frameworks is that in LF the theorem prover can take advantage of the inherent structure of the objects which can be quite complex since they typically involve higher-order representation techniques together with dependent types. Other advantages include efficient encodings of a variety of deductive systems, automatic availability of various weakening and substitution lemmas for the encodings, and a limited set of meta-logical operators suitable for proof search. Those operators include the application of induction hypotheses, case analysis over universally bound assumptions, and straightforward resolution theorem proving to search for witness objects of existentially quantified assumptions.

Theorem proving in a logical framework setting suffers from the same combinatorial problems other theorem prover also suffer from. One natural question to ask is, how the proof planning techniques developed for the first-order domain can be carried over to the domain of higher-order logical frameworks. In particular, how does proof planning techniques interact with our approach to theorem proving that follows the philosophy of Nqthm [BM79]? A theoretical development consists of a sequence of lemmas which can all be proven automatically. If the gap in-between two lemmas is too large, the theory implementer has to introduce auxiliary lemmas.

One of the main problems in automated theorem proving in a logical framework such as Twelf is the choice of a universally bound variable for case analysis, also called splitting candidate. Usually, the set of splitting candidates grows quite big even for simple theorem proving problems. In addition, previous attempts to limit the set of candidates by considering only those assumptions that do not occur as index elsewhere, were incomplete. Other heuristics have been successfully applied on a small scale, but have failed to scale to bigger examples.

The key to success is recognizing early when a prover has reached an unsuccessful state. To this end, we develop in this paper a proof planning calculus $\mathcal{P}_\omega^+$ which captures the flow of information when applying lemmas, induction hypotheses, or simply inference rules. Searching for proof plans serves as a criterion for the prospects of a proof state to lead eventually to a proof of the theorem. In addition, proof plans contain plenty of information about which resources have been used and others that have not. In future work we plan to exploit this information for constructing the actual proof.

$\mathcal{P}^+_\omega$ is a natural deduction calculus for a fragment of first-order intuitionistic monadic logic and hence decidable. Finding a proof plan is straightforward. First, a proof state is converted into a $\mathcal{P}^+_\omega$-formula, which is passed to a resolution theorem prover. If a proof plan for this formula cannot be found in $\mathcal{P}^+_\omega$ then the original proof state cannot be proven in $\mathcal{M}^+_\omega$. Our meta theory guarantees that $\mathcal{P}^+_\omega$ is sound. If a formula is provable in $\mathcal{M}^+_\omega$, $\mathcal{P}^+_\omega$ will be able to construct an appropriate proof plan.

The paper is organized as follows. We first discuss the type preservation theorem for Mini-ML as running example in Section 2. We then give a brief introduction to the meta-logic $\mathcal{M}^+_\omega$ in Section 3 before we define the proof planning calculus $\mathcal{P}^+_\omega$ in Section 4 and its meta theory in Section 5. Finally we conclude and assess results in Section 6.

2 Example

One of the examples that demonstrates our techniques well is the type preservation theorem of a simply typed functional language. Let us consider a fragment of Mini-ML [MP91] that contains only constructs for λ-abstraction "lam", application "app", and fixed points "fix". Although this language is quite impoverished, it illustrates well the techniques described in this paper. In fact, our techniques scale to full Mini-ML. Also, in the interest of space, we restrict the presentation to a version of the language already represented in the logical framework LF whose formal presentation we postpone until Section 3. The reader not familiar with Mini-ML is referred to [MP91] for a detailed description. Expressions in Mini-ML are represented as objects of type "exp".

exp : type.
lam : (exp → exp) → exp.
app : exp → exp → exp.
fix : (exp → exp) → exp.

Note the use of higher-order abstract syntax for "lam" and "fix" as a mechanism to bind Mini-ML level variables by variables of LF. Due to the simplicity of the fragment, Mini-ML provides only one type constructor "arrow".

tp : type.
arrow : tp → tp → tp.

Under the judgments as types paradigm, the typing judgment $\Gamma \vdash e : \tau$ is expressed as a type family indexed by an expression and a type, that is an object of type exp, and tp, respectively. The context Γ itself is represented by the LF context.

of : exp → tp → type.
tp_lam : (Πx:exp. of x T_1 → of (E x) T_2)
→ of (lam E) (arrow T_1 T_2).
tp_app : of E_2 T_2 → of E_1 (arrow T_2 T_1)

```
          → of (app E1 E2) T1.
tp_fix : (Πx:exp. of x T → of (E x) T)
          → of (fix E) T.
```

Hypothetical judgments that add new hypotheses to Γ are encoded using higher-order techniques, as the functional argument to tp_lam shows. Following standard practice [Pfe91] we omit all implicit Π-abstractions from types and we take $\beta\eta$-conversion as the notion of definitional equality [Coq91].

Finally, we give an operational semantics for this language which is a standard and call-by-value.

```
eval : exp → exp → type.
ev_lam : eval (lam E) (lam E).
ev_app : eval (E1' V2) V → eval E2 V2 → eval E1 (lam E1')
          → eval (app E1 E2) V.
ev_fix : eval (E (fix E)) V
          → eval (fix E) V.
```

It is important to point out how efficient the representation of the evaluation application rule is. Mini-ML programs in LF are considered equivalent modulo α-conversion. The substitution lemma that occurs in an informal proof is expressed by a simple LF level application $(E_1'\ V_2)$ in the definition of ev_app.

The meta-logical foundation for reasoning about deductive systems that are encoded in LF is discussed in detail in [Sch00]. The underlying concept of the meta-theorem prover implemented in the Twelf system is that of a proof state, which consists of a set of assumptions and a goal. The type preservation theorem, for example,

(1) $\quad \forall E : \text{exp}.\, \forall V : \text{exp}.\, \forall T : \text{tp}.\, \forall D : \text{eval}\ E\ V.\, \forall P : \text{of}\ E\ T.\, \exists Q : \text{of}\ V\ T.\, \top$

leads to an initial state of the following form.

$$E : \text{exp}, V : \text{exp}, T : \text{tp}, D : \text{eval}\ E\ V, P : \text{of}\ E\ T \vdash \exists Q : \text{of}\ V\ T.\, \top$$

To the left of the $\vdash$ symbol you find all assumptions that are made available to prove the theorem, to its right the formula that is to be proven. To simplify notation, we will use Ψ to refer to the list of assumptions. A complete proof consists of a recipe of how to construct a Q out of E, V, T, D and P, and constants defined in the LF signature.

The reasoning process can be described by the following three fundamental operations: *Splitting* analyzes cases over variables declared in Ψ, such as D : eval $E\ V$ and results in general in a set of new proof states. *Recursion* applies induction hypotheses and inserts new assumptions into Ψ. And finally, *filling* attempts to construct LF objects for a given LF type by trying to generate witness objects for existential quantifiers on the right, or argument objects for universal quantifiers on the left of the $\vdash$ symbol. All operations are nondeterministic.

One of the main contributions of this paper is a proof planner that maps out a way on how to close a proof state. The original problem space is simplified by

converting proof states into proof planning problems which are formulated as theorem proving problems in first-order intuitionistic logic. The corresponding proof calculus is a natural deduction calculus. Therefore, the task of proof planning is in fact executed by a theorem prover, and consequently a proof plan is nothing but a natural deduction derivation. Our meta-theoretic considerations guarantee that if the current proof state can be closed the proof planner is guaranteed to find a plan.

When our proof planning algorithm is in action, it recursively constructs a complete proof plan by continuously splitting assumptions, applying induction hypotheses, and considering all lemmas. Once a plan is successfully constructed, the theorem prover is invoked possibly taking advantage of the information contained in the plan. The plan may shed some light on what variable to split, which induction hypothesis or lemma to apply next, and possibly even limits the fragment of the deductive system in which to conduct proof search. This is a tremendous advancement compared to the naive and straightforward implementation of the meta theorem prover implemented in Twelf.

The subterm relationship between objects and parameters that underlies the proof planning technique presented in this paper is the notion of *approximate typability*. Intuitively, approximate typability is a weak form of typing. Instead of saying that M has type of $E\ V$, we record only the information that object M contains information about objects E and V which we express as approximate type $(\text{of } E) \wedge (\text{of } V)$. Clearly, by moving to approximate types we loose a lot of information about the objects themselves. On the other hand, we gain the freedom to look for proof plans i.e. natural deduction proofs among approximate types. The main theoretical result of this paper states, that if for a particular theorem proving problem no proof plan can be found, a direct proof is guaranteed not to exist. If it exists, however, we can use information gained from the proof plan to guide the theorem prover. Following the proposed classifications of Giunchiglia and Walsh, this means that our notion of approximation satisfies the properties of a TI abstraction [GW92] with a consistent abstract space. In order to illustrate the proof planning technique we take as an example the proof state for the type preservation theorem and analyze cases over $D : \text{eval } E\ V$ yielding three cases.

1. $E' : \text{exp} \to \text{exp}, T : \text{tp}, P : \text{of } (\text{lam } E')\ T$

   $\vdash \exists Q : \text{of } (\text{lam } E')\ T.\ \top$

2. $E_1 : \text{exp}, E_1' : \text{exp} \to \text{exp}, E_2 : \text{exp}, V_2 : \text{exp}, V : \text{exp}, T : \text{tp},$

   $D_1 : \text{eval } E_1\ (\text{lam } E_1'), D_2 : \text{eval } E_2\ V_2, D_3 : \text{eval } (E_1'\ V_2)\ V,$

   $P : \text{of } (\text{app } E_1\ E_2)\ T$

   $\vdash \exists Q : \text{of } V\ T.\ \top$

3. $E' : \exp \to \exp, V : \exp, T : \mathrm{tp}, D : \mathrm{eval}\ (E'\ (\mathrm{fix}\ E'))\ V, P : \mathrm{of}\ (\mathrm{fix}\ E')\ T$
   $\vdash \exists Q : \mathrm{of}\ V\ T. \top$

and after computing all applicable induction hypotheses, we obtain:

1. $E' : \exp \to \exp, T : \mathrm{tp}, P : \mathrm{of}\ (\mathrm{lam}\ E')\ T$
   $\vdash \exists Q : \mathrm{of}\ (\mathrm{lam}\ E')\ T. \top$

2. $E_1 : \exp, E_1' : \exp \to \exp, E_2 : \exp, V_2 : \exp, V : \exp, T : \mathrm{tp},$
   $D_1 : \mathrm{eval}\ E_1\ (\mathrm{lam}\ E_1'), D_2 : \mathrm{eval}\ E_2\ V_2, D_3 : \mathrm{eval}\ (E_1'\ V_2)\ V,$
   $P : \mathrm{of}\ (\mathrm{app}\ E_1\ E_2)\ T,$
   $\mathbf{ih}_1 \in \forall t : \mathrm{tp}. \forall u : \mathrm{of}\ E_1\ \mathrm{t}. \exists q : \mathrm{of}\ (\mathrm{lam}\ E_1')\ \mathrm{t}. \top,$
   $\mathbf{ih}_2 \in \forall t : \mathrm{tp}. \forall u : \mathrm{of}\ E_2\ \mathrm{t}. \exists q : \mathrm{of}\ V_2\ \mathrm{t}. \top,$
   $\mathbf{ih}_3 \in \forall t : \mathrm{tp}. \forall u : \mathrm{of}\ (E_1'\ V_2)\ \mathrm{t}. \exists q : \mathrm{of}\ V\ \mathrm{t}. \top,$
   $\vdash \exists Q : \mathrm{of}\ V\ T. \top$

3. $E' : \exp \to \exp, V : \exp, T : \mathrm{tp}, D : \mathrm{eval}\ (E'\ (\mathrm{fix}\ E'))\ V, P : \mathrm{of}\ (\mathrm{fix}\ E')\ T,$
   $\mathbf{ih} \in \forall t : \mathrm{tp}. \forall p : \mathrm{of}\ (E'\ (\mathrm{fix}\ E'))\ t. \exists q : \mathrm{of}\ V\ t. \top$
   $\vdash \exists Q : \mathrm{of}\ V\ T. \top$

Proof planning with the approximate types in each of the three cases yields the following observations: The goal formula of the first state corresponds to the approximate type $(\mathrm{of}\ E') \wedge (\mathrm{of}\ T)$ from Q, which is immediately discharged by the approximate type of P. The goal of the second state corresponds to the approximate type $(\mathrm{of}\ V) \wedge (\mathrm{of}\ T)$. Using the approximate type of q in $\mathbf{ih}_3$ after instantiating t with T yields $(\mathrm{of}\ E_1') \wedge (\mathrm{of}\ V_2) \wedge (\mathrm{of}\ T)$. This can be refined by $\mathbf{ih}_1$ and $\mathbf{ih}_2$ to $(\mathrm{of}\ E_1) \wedge (\mathrm{of}\ E_2) \wedge (\mathrm{of}\ T)$, which is discharged by the approximate type of P. The goal of the third state corresponds to the approximate type $(\mathrm{of}\ V) \wedge (\mathrm{of}\ T)$. Using the approximate type of q in $\mathbf{ih}$, it entails $(\mathrm{of}\ E') \wedge (\mathrm{of}\ T)$ which can be immediately discharged by the approximate type of P.

This proof plan exhibits that each case may be directly provable, and therefore the meta-theorem prover should try to follow the plan taking advantage of the information gathered throughout the planning process. Consider for example the second case above when analyzing cases over D: The derivation of the approximate goal contained backward applications of $\mathbf{ih}_3$, $\mathbf{ih}_1$, $\mathbf{ih}_2$, and finally P. This information can be used by the filling operation to construct an object of type $Q : \mathrm{of}\ V\ T$. In general, however, a planner must provide all possible proof plans for the meta theorem prover to extract information which rules to use during filling.

$$
\begin{array}{lll}
\text{Kinds:} & K & ::= \text{type} \mid \Pi x : A.\, K \\
\text{Types:} & A & ::= a \mid A\ M \mid \Pi x : A_1.\, A_2 \\
\text{Objects:} & M & ::= c \mid x \mid \lambda x : A.\, M \mid M_1\ M_2 \mid \pi_x(\underline{y}) \\
\text{Worlds:} & \Phi & ::= L : \text{some } \Gamma_1 \text{ block } \Gamma_2 \mid \Phi + \Phi \mid \Phi^* \\
\text{Signatures:} & \Sigma & ::= \cdot \mid \Sigma, c : A \mid \Sigma, a : K \\
\text{Substitutions:} & \sigma & ::= \text{id} \mid \sigma, M/x \mid \sigma, \underline{y}/\underline{x} \\
\text{Formulas:} & F & ::= \forall x : A.\, F \mid \forall \underline{x} : (L; \sigma).\, F \mid F_1 \supset F_2 \\
 & & \quad \mid \exists x : A.\, F \mid \exists \underline{x} : (L; \sigma).\, F \mid F_1 \wedge F_2 \mid \top \\
\text{Programs:} & P & ::= \Lambda x : A.P \mid \Lambda \underline{x} : (L; \sigma).P \mid \Lambda \mathbf{x} \in F.P \\
 & & \quad \mid \mathbf{x} \mid \langle M; P \rangle \mid \langle \underline{x}; P \rangle \mid \langle P_1; P_2 \rangle \mid \langle\rangle \mid P\ M \\
 & & \quad \mid \nu P \mid P \underline{x} \mid P_1\ P_2 \mid \texttt{case}\ \Omega \mid \mu \mathbf{x} \in F.P \\
\text{Cases:} & \Omega & ::= . \mid \Omega, (\Psi \rhd \sigma \mapsto P) \\
\text{Contexts:} & \Psi & ::= \cdot \mid \Psi, x : A \mid \Psi, \underline{x} : (L; \sigma) \mid \Psi, \mathbf{x} \in F
\end{array}
$$

Fig. 1. Syntactic Categories for $\mathcal{M}_\omega^+$

In fact, the plan above will eventually lead to success. It is an easy exercise to determine, that the sketched planning algorithm will not find a proof plan when considering cases solely over V or E. Intuitively, this is to be expected, since the proof of the type preservation theorem goes either by induction on D or P, but never on V or E.

3 The Meta Logic $\mathcal{M}_\omega^+$

In this work, the type theory that serves the representation of deductive systems is the logical framework LF [HHP93]. It has been employed for representing type systems, operational semantics, logics, and in particular Church's higher-order logic, which is used in research about secure mobile code [App01] and proof carrying code [Nec97]. Figure 2 shows the usual definition of LF as a three layered system consisting of objects M, types A, and kinds K. We slightly deviate from the standard notation and present a version of LF which includes block variables [Sch00,Sch01a,Sch01b]. Block variables $\underline{y}$ are underlined and range over instances of a world Φ. We write Γ for those contexts Ψ that contain only declarations of the form $x : A$, and $\underline{x} : (L; \sigma)$. They correspond to the standard contexts of LF. LF signatures denoted by Σ are

standard, and substitutions σ differ from standard substitutions in that they admit the renaming of block variables.

$$\frac{\Sigma(c) = A}{\Gamma \vdash_{\Sigma;\Phi} c : A}\ \mathsf{obj\_con} \qquad \frac{\Gamma(x) = A}{\Gamma \vdash_{\Sigma;\Phi} x : A}\ \mathsf{obj\_var} \qquad \frac{\Gamma(\underline{y}) = (L;\sigma) \quad (\text{block } L)[\sigma](x) = A}{\Gamma \vdash_{\Sigma;\Phi} \pi_x(\underline{y}) : A}\ \mathsf{obj\_proj}$$

$$\frac{\Gamma \vdash_{\Sigma;\Phi} A_1 : \text{type} \quad \Gamma, x : A_1 \vdash_{\Sigma;\Phi} M : A_2}{\Gamma \vdash_{\Sigma;\Phi} \lambda x : A_1.\, M : \Pi x : A_1.\, A_2}\ \mathsf{obj\_lam}$$

$$\frac{\Gamma \vdash_{\Sigma;\Phi} M_1 : \Pi x : A_2.\, A_1 \quad \Gamma \vdash_{\Sigma;\Phi} M_2 : A_2}{\Gamma \vdash_{\Sigma;\Phi} M_1\ M_2 : A_1[M_2/x]}\ \mathsf{obj\_app}$$

$$\frac{\Sigma(a) = K}{\Gamma \vdash_{\Sigma;\Phi} a : K}\ \mathsf{tp\_const} \qquad \frac{\Gamma \vdash_{\Sigma;\Phi} A_1 : \text{type} \quad \Gamma, x : A_1 \vdash_{\Sigma;\Phi} A_2 : \text{type}}{\Gamma \vdash_{\Sigma;\Phi} \Pi x : A_1.\, A_2 : \text{type}}\ \mathsf{tp\_pi}$$

$$\frac{\Gamma \vdash_{\Sigma;\Phi} A_1 : \Pi x : A_2.\, K \quad \Gamma \vdash_{\Sigma;\Phi} M : A_2}{\Gamma \vdash_{\Sigma;\Phi} A_1\ M : K[M/x]}\ \mathsf{tp\_app}$$

$$\frac{}{\Gamma \vdash_{\Sigma;\Phi} \text{type} : \text{ kind}}\ \mathsf{kd\_type} \qquad \frac{\Gamma \vdash_{\Sigma;\Phi} A : \text{type} \quad \Gamma, x : A \vdash_{\Sigma;\Phi} K : \text{ kind}}{\Gamma \vdash_{\Sigma;\Phi} \Pi x : A.K \text{ kind}}\ \mathsf{kd\_pi}$$

Fig. 2. The Logical Framework LF.

Projections from block variables are written as $\pi_x(\underline{y})$. They must adhere to the structure of the block corresponding to $\underline{y} : (L;\sigma)$. This means that x must be declared in the block part Γ_2 of the block declaration of "L : some Γ_1 block Γ_2" in Φ. The substitution σ simply instantiates all variables in Γ_1 which may occur free in Γ_2. We also write $(\text{block } L)[\sigma](x)$ to refer to the type of parameter x.

Block variables can be seen as dynamic extensions of the set of constants defining a type. Since theorem proving in $\mathcal{M}_\omega^+$ takes place in a higher-order setting [Sch01b], appeals to the induction hypothesis may have to traverse λ-binders, prompting the theorem prover to introduce new parameters in form of block variables in accordance with the world declaration.

Intuitively, worlds Φ resemble regular expressions, whose language is a set of regularly formed LF contexts. Each declaration in any of those LF contexts can be thought of as a new constructor, dynamically extending the types already defined in the signature. Without loss of generality, we can assume all LF objects, LF types, and LF kinds to be in canonical form. The typing rules for our version of LF are depicted in Figure 2.

The meta logic $\mathcal{M}_\omega^+$ that is designed for reasoning about LF encodings is defined on a different level from LF. Due to the higher-order nature of the encodings it is impossible to identify the quantifiers for LF with the quantifiers of the meta logic [Sch01b]. On the contrary, both levels, LF and $\mathcal{M}_\omega^+$ must be carefully kept apart. The distinguishing characteristics between the two levels are that $\mathcal{M}_\omega^+$'s programs correspond to functions that are definable

$$\frac{\Psi \vdash_{\Sigma;\Phi} A : \text{type} \quad \Psi, x : A \vdash_{\Sigma;\Phi} P \in F}{\Psi \vdash_{\Sigma;\Phi} \Lambda x : A.\, P \in \forall x : A.\, F}\ \forall\text{I\_LF} \qquad \frac{\Psi \vdash_{\Sigma;\Phi} P \in \forall x : A.\, F \quad \Psi \vdash_{\Sigma} M : A}{\Psi \vdash_{\Sigma;\Phi} P\ M \in F[M/x]}\ \forall\text{E\_LF}$$

$$\frac{\Psi, \underline{x} : (L;\sigma) \vdash_{\Sigma;\Phi} P \in F}{\Psi \vdash_{\Sigma;\Phi} \Lambda \underline{x} : (L;\sigma).\, P \in \forall \underline{x} : (L;\sigma).\, F}\ \forall\text{I\_block} \qquad \frac{\Psi \vdash_{\Sigma;\Phi} P \in \forall \underline{x} : (L;\sigma).\, F \quad \Psi(\underline{y}) = (L;\sigma)}{\Psi \vdash_{\Sigma;\Phi} P\ \underline{y} \in F[\underline{y}/\underline{x}]}\ \forall\text{E\_block}$$

$$\frac{\Psi, \mathbf{x} \in F_1 \vdash_{\Sigma;\Phi} P \in F_2}{\Psi \vdash_{\Sigma;\Phi} \Lambda \mathbf{x} \in F_1.\, P \in F_1 \supset F_2}\ \forall\text{I\_meta} \qquad \frac{\Psi \vdash_{\Sigma;\Phi} P_1 \in F_2 \supset F_1 \quad \Psi \vdash_{\Sigma;\Phi} P_2 \in F_2}{\Psi \vdash_{\Sigma;\Phi} P_1\ P_2 \in F_1}\ \forall\text{E\_meta}$$

$$\frac{\Psi \vdash_{\Sigma} M : A \quad \Psi \vdash_{\Sigma;\Phi} P \in F[M/x]}{\Psi \vdash_{\Sigma;\Phi} \langle M; P\rangle \in \exists x : A.\, F}\ \exists\text{I\_LF} \qquad \frac{\Psi(\underline{y}) = (L;\sigma) \quad \Psi \vdash_{\Sigma;\Phi} P \in F[\underline{y}/\underline{x}]}{\Psi \vdash_{\Sigma;\Phi} \langle \underline{y}; P\rangle \in \exists \underline{x} : (L;\sigma).F}\ \exists\text{I\_block}$$

$$\frac{\Psi \vdash_{\Sigma;\Phi} P_1 \in F_1 \quad \Psi \vdash_{\Sigma;\Phi} P_2 \in F_2}{\Psi \vdash_{\Sigma;\Phi} \langle P_1; P_2\rangle \in F_1 \wedge F_2}\ \wedge\text{I} \qquad \frac{\Psi(\mathbf{x}) = F}{\Psi \vdash_{\Sigma;\Phi} \mathbf{x} \in F}\ \text{axiom} \qquad \frac{}{\Psi \vdash_{\Sigma;\Phi} \langle\rangle \in \top}\ \top\text{I}$$

$$\frac{\Psi, \mathbf{x} \in F \vdash_{\Sigma;\Phi} P \in F}{\Psi \vdash_{\Sigma;\Phi} \mu \mathbf{x} \in F.\, P \in F}\ \text{rec}^1 \qquad \frac{\Psi \vdash_{\Sigma;\Phi} \Omega \in F}{\Psi \vdash_{\Sigma;\Phi} \text{case}\ \Omega \in F}\ \text{case}^2$$

$$\frac{\Psi \vdash_{\Sigma;\Phi} P_1 \in F_1 \quad \Psi, \mathbf{x} \in F_1 \vdash_{\Sigma;\Phi} P_2 \in F_2}{\Psi \vdash_{\Sigma;\Phi} \text{let}\ \mathbf{x} = P_1\ \text{in}\ P_2 \in F_2}\ \text{let}$$

$$\frac{\Psi \vdash_{\Sigma;\Phi} P \in \forall \underline{x} : (L;\sigma).\, F \quad \text{abs}\,((\text{block}\ L)[\sigma]).\, F = F'}{\Psi \vdash_{\sigma;\Phi} \nu P \in F'}\ \text{new}$$

$$\frac{}{\Psi \vdash_{\Sigma;\Phi} \cdot \in F}\ \text{empty} \qquad \frac{\Psi_1 \vdash_{\Sigma;\Phi} \Omega \in F \quad \Psi_2 \vdash_{\Sigma;\Phi} \sigma \in \Psi_1 \quad \Psi_2 \vdash_{\Sigma;\Phi} P \in F[\sigma]}{\Psi_1 \vdash_{\Sigma;\Phi} \Omega, (\Psi_2 \triangleright \sigma \mapsto P) \in F}\ \text{cons}$$

Fig. 3. Meta Logic $\mathcal{M}^+_\omega$.

through pattern matching for which no canonical forms are known to exist. LF functions, on the other hand, do not allow function definition by cases. It relies crucially on the existence of canonical forms otherwise it could not guarantee adequate representations.

$\mathcal{M}^+_\omega$'s features include proof by induction over arbitrary higher-order encodings, i.e. encodings that make use of negative occurrences of parameter binders with automatic application of substitution, weakening, and exchange lemmas. The syntactic categories provided by the meta-logic $\mathcal{M}^+_\omega$ are depicted in the lower half of Figure 1. F stands for the first-order formulas that express the properties to be proven. For example, the type preservation theorem in Equation (1) is an $\mathcal{M}^+_\omega$ formula. Universal and existential quantifiers may range over LF variables and block variables. Implication and conjunction are standard. Compared with other logics, $\mathcal{M}^+_\omega$ lacks user defined relational constants, falsehood, and disjunction. We are currently investigating how those operators can be added to $\mathcal{M}^+_\omega$.

Programs P summarize proof derivations, but they are of little interest for this work. We present them here anyway because they play an important role

in the soundness proof of $\mathcal{M}^+_\omega$. Under a realizer interpretation those proof terms act as total functions mapping instantiations of universal quantifiers to instantiations of existentials. $\mathcal{M}^+_\omega$ distinguishes three different kinds of variables. LF variables $x : A$, block variables $\underline{x} : (L; \sigma)$, and meta variables $\mathbf{x} \in F$. We overload Λ to bind variables of all three kinds, and $\langle \cdot; \cdot \rangle$ for pairs of LF objects, blocks, or programs with programs, respectively. $\langle\rangle$ stands for unit, and juxtaposition for application. We distinguish three different kinds of application $P\ M$, $P\ \underline{x}$, and $P_1\ P_2$. ν is the program that introduces new constants dynamically and interprets the result as functional objects in LF. μ and "case" are the two program constructors that implement recursion, and case analysis, respectively. "case" expects as argument a list of cases Ω, which may be empty. Patterns are represented as substitutions σ, defined in a new environment Ψ, and P is the body of the pattern.

The proof rules of $\mathcal{M}^+_\omega$ are given in Figure 3. Many of the rules are standard, a few however require explanation. The rule new introduces new parameter blocks into the context, and the result of executing the body is later abstracted to the LF level [Sch01b]. The technique is based on the idea to represent hypothetical derivations as parametric functions in LF; we generalize the technique to $\mathcal{M}^+_\omega$ formulas, and write abs $((\text{block } L)[\sigma]).\, F = F'$ where F' is the result of abstracting each LF declaration in F in turn. In a slight deviation from previous presentations of $\mathcal{M}^+_\omega$ we split the introduction of new parameters by $\forall$I_block from their abstraction in new which leads to a cleaner presentation of the various concepts. Nevertheless, in most $\mathcal{M}^+_\omega$ derivations, applications of $\forall$I_block are immediately followed by an application of the new rule.

What makes $\mathcal{M}^+_\omega$ a sound meta-logic is that all programs are total which is enforced by two side conditions. The first is associated with rec, denoted by ^{1} and requires that $\mu\mathbf{x}.\, P$ is terminating. The second is associated with case, denoted by ^{2}, and requires that all cases are covered. For each environment η that instantiates all variables in Ψ, (for which we use the standard substitution notation $\Gamma \vdash \eta \in \Psi$), Ω must contain a case $(\Psi' \triangleright \sigma \mapsto P)$ that matches it. We say that σ matches η, if η can be written as a substitution composition of a new environment η' $(\Gamma \vdash \eta' : \Psi')$ and σ.

In previous work [Sch00], the first author has given syntactic criteria for the implementation of the two side conditions, which have led to the meta logic $\mathcal{M}^+_2$. The 2 refers to the fragment of $\mathcal{M}^+_\omega$ with only Π_2-formulas, these are formulas with only one quantifier alternation. The techniques developed in this paper however scale beyond the Π_2-fragment. In fact, any syntactic criteria for the two side conditions that guarantee totality are sufficient. We therefore concentrate the current research on $\mathcal{M}^+_\omega$ and develop the proof planning calculus $\mathcal{P}^+_\omega$ in the next section.

$$\text{Approximation Formulas: } G ::= \forall x.\, G \mid \forall \underline{x} : L.\, G \mid G_1 \supset G_2 \mid \exists x.\, G \mid \exists \underline{x} : L.\, G \mid G_1 \wedge G_2 \mid \top \mid (a\ x)$$

$$\text{Approximation Contexts: } \Delta ::= \cdot \mid \Delta, G \mid \Delta, x \mid \Delta, \underline{x} : L$$

Fig. 4. Syntactic Categories for $\mathcal{P}^+_\omega$.

4 The Proof Planning Calculus $\mathcal{P}^+_\omega$

Our approach to proof planning is purely logical. This is exhibited by the fact that the here proposed proof planning calculus $\mathcal{P}^+_\omega$ is in essence a natural deduction calculus. Any valid derivation in $\mathcal{P}^+_\omega$ is called a proof plan. As discussed in Section 2, proof plans contain information that may guide the meta theorem prover to close a branch in a proof. For example, the set of proof plans for a particular goal contains information about which lemmas, and which induction hypotheses to apply, even which constants are necessary to construct witness objects in LF, and potentially information about the order in which constructors are to be applied.

Therefore, the task of constructing a proof plan reduces to finding a derivation in $\mathcal{P}^+_\omega$. $\mathcal{P}^+_\omega$ lies in a fragment of first-order intuitionistic monadic logic, which is decidable [Tho90]. Consequently effective decision procedures exist. Resolution and tableaux methods are only a few techniques that are applicable in a proof planner whose description we leave to a future paper.

Any proof state of the meta theorem prover that searches for derivations in $\mathcal{M}^+_\omega$ can be translated into a proof planning problem as motivated already in Section 2. Hereby we translate the proof state via the process of *approximation*. The goal formula is translated into a $\mathcal{P}^+_\omega$ formula, which we also call *approximation formula*, and the proof context into a $\mathcal{P}^+_\omega$ context, which is also called *approximation context*. Both syntactic categories are depicted in Figure 4. Approximation is a form of abstraction [GW92].

When comparing $\mathcal{M}^+_\omega$ formulas and $\mathcal{M}^+_\omega$ contexts with $\mathcal{P}^+_\omega$ formulas and $\mathcal{P}^+_\omega$ contexts, respectively, we observe that they are very similar. This is not very surprising, because the main difference between both calculi lies in the treatment of LF related assumptions. In $\mathcal{P}^+_\omega$, for example, LF type labels are entirely eliminated from the quantifiers and replaced instead by an approximation formula, that often consists of a conjunction of several propositions $(a\ x)$. a is a type family, and x a variable that may occur as a subobject in an object of type family a. Besides approximation formulas, approximation contexts contain also variable names x, and block variable declarations $\underline{x} : L$. The translation operation from $\mathcal{M}^+_\omega$ to $\mathcal{P}^+_\omega$ is described in detail in Section 4.1. In Section 4.3 we then introduce the calculus for $\mathcal{P}^+_\omega$.

$$\frac{}{\Psi;\Gamma \vdash a' \rightsquigarrow_a \top}\ \mathsf{atom\_tp\_fam}$$

$$\frac{\Psi;\Gamma \vdash A_1 \rightsquigarrow_a G_1 \qquad \Psi;\Gamma, x : A_1 \vdash A_2 \rightsquigarrow_a G_2}{\Psi;\Gamma \vdash \Pi x : A_1.\, A_2 \rightsquigarrow_a G_1 \wedge G_2}\ \mathsf{atom\_tp\_pi}$$

$$\frac{\Psi;\Gamma \vdash A \rightsquigarrow_a G_1 \qquad \Psi;\Gamma \vdash M \rightsquigarrow_a G_2}{\Psi;\Gamma \vdash A\ M \rightsquigarrow_a G_1 \wedge G_2}\ \mathsf{atom\_tp\_app}$$

$$\frac{}{\Psi;\Gamma \vdash c \rightsquigarrow_a \top}\ \mathsf{atom\_obj\_const}$$

$$\frac{\Gamma(x) = A}{\Psi;\Gamma \vdash x \rightsquigarrow_a \top}\ \mathsf{atom\_obj\_par} \qquad \frac{\Psi(x) = A}{\Psi;\Gamma \vdash x \rightsquigarrow_a (a\ x)}\ \mathsf{atom\_obj\_var}$$

$$\frac{}{\Psi;\Gamma \vdash \pi_x(\underline{y}) \rightsquigarrow_a (a\ \pi_x(\underline{y}))}\ \mathsf{atom\_obj\_proj}$$

$$\frac{\Psi;\Gamma \vdash A \rightsquigarrow_a G_1 \qquad \Psi;\Gamma, x : A \vdash M \rightsquigarrow_a G_2}{\Psi;\Gamma \vdash \lambda x : A.\, M \rightsquigarrow_a G_1 \wedge G_2}\ \mathsf{atom\_obj\_lam}$$

$$\frac{\Psi;\Gamma \vdash M_1 \rightsquigarrow_a G_1 \qquad \Psi;\Gamma \vdash M_2 \rightsquigarrow_a G_2}{\Psi;\Gamma \vdash M_1\ M_2 \rightsquigarrow_a G_1 \wedge G_2}\ \mathsf{atom\_obj\_app}$$

Fig. 5. Translation of Atomic Types and Objects.

4.1 Translation from $\mathcal{M}^+_\omega$ to $\mathcal{P}^+_\omega$

We begin now with the formal definition of the translation relation. The translation relation is defined on $\mathcal{M}^+_\omega$ formulas, types, objects, and contexts via the following six translation judgments:

Translation of types:	$\Psi \vdash A \rightsquigarrow_a G$
Translation of objects:	$\Psi \vdash M \rightsquigarrow_a G$
Translation of substitution lemmas:	$\Psi \vdash A \rightsquigarrow G$
Translation of induction hypothesis:	$\Psi \vdash F \rightsquigarrow G$
Translation of contexts:	$\vdash \Psi \rightsquigarrow \Delta$
Translation of blocks:	$\Psi \vdash \Gamma \rightsquigarrow \Delta; \sigma$

Each of these judgments is explained below in turn.

4.1.1 Types and objects.

The rules that define the first two judgments are given in Figure 5. Assume that M, or A, respectively occur as subobjects in the indices to a type family a. The result of transforming A or M is an approximation formula that captures the necessary subobject relation between parameters, variables, and projections, and the type family a. As an example, let P : of (app E_1 E_2) T

in (2) be a parameter in Ψ. After a few simplifications we obtain

$$\Psi \vdash \text{of } (\text{app } E_1\ E_2)\ T \rightsquigarrow_{\text{of}} (\text{of } E_1) \wedge (\text{of } E_2) \wedge (\text{of } T)$$

which reads informally as that objects of type family "of" contain information about E_1, E_2, and T. The conversion to approximation formulas, however, does not preserve locality information among assumptions.

4.1.2 Substitution lemmas and induction hypotheses.
In $\mathcal{M}^+_\omega$, a proof context may contain hypotheses of functional LF type. When such a hypothesis is applied to arguments it will reduce to a normal form; just in the same way as if one applies a substitution lemma. Hypothesis of this kind are not at all uncommon in our setting. For example one more case analysis on P in the third case of the type preservation example from Section 2 makes such a functional hypothesis P' available.

$$\begin{aligned} 3'.\ & E' : \text{exp} \to \text{exp}, V : \text{exp}, T_1 : \text{tp}, T_2 : \text{tp}, D : \text{eval } (E'\ (\text{fix } E'))\ V, \\ & \textbf{ih} \in \forall t : \text{tp}.\, \forall p : \text{of } (E'\ (\text{fix } E'))\ t.\, \exists q : \text{of } V\ t.\, \top, \\ & P' : \Pi x : \text{exp}.\, \text{of } x\ T_1 \to \text{of } (E\ x)\ T_2 \\ & \quad \vdash \exists Q : \text{of } V\ (\text{arrow } T_1\ T_2).\, \top \end{aligned}$$

In $\mathcal{P}^+_\omega$, substitution lemmas must be translated the same way as induction hypotheses are translated: As rules that can be applied to reason about the flow of information. For example, if **ih** is applied to T then $(\text{of } E') \wedge (\text{of } T) \supset (\text{of } V) \wedge (\text{of } T)$. The proof planning rules to be presented in Section 4.3 of $\mathcal{P}^+_\omega$ permit reasoning about the flow of information as part of the proof plan. The translation relation for substitution lemmas and induction hypotheses are defined by the third and fourth judgment, respectively, with the rules that are given in Figure 6. In those rules we use the notation $a = \text{head } A$, that determines the head of an atomic type A. It denotes the type family A belongs to.

$$\text{head } A = \begin{cases} a & \text{if } A = a \\ \text{head } A' & \text{if } A = A'\ M \end{cases}$$

For example the head of type "of (app M_1 M_2) T" is the type family "of".

When translating universally quantified formulas over blocks such as the one in trans_allP, one first transforms the context (block $L)[\sigma]$ into an approximation context Δ following the rules defined in Figure 7 explained below. Δ is then combined with the result of transforming F. We use the shorthand $\Delta \supset G$ for the resulting approximation formula.

$$\Delta \supset G = \begin{cases} G & \text{if } \Delta = \cdot \\ \Delta' \supset (G' \supset G) & \text{if } \Delta = \Delta', G' \end{cases}$$

An existential quantified formula such as the one described in trans_exP is translated analogously, except that all assumptions in Δ are glued together

$$\frac{\Psi \vdash A \rightsquigarrow_a G}{\Psi \vdash A \rightsquigarrow G}\ \mathsf{trans\_atom} \qquad \text{if } A \text{ is atomic and } a = \text{head } A$$

$$\frac{\Psi \vdash A_1 \rightsquigarrow G_1 \qquad \Psi, x : A_1 \vdash A_2 \rightsquigarrow G_2}{\Psi \vdash \Pi x : A_1.\, A_2 \rightsquigarrow G_1 \supset \forall x.\, G_2}\ \mathsf{trans\_pi}$$

$$\frac{}{\Psi \vdash \top \rightsquigarrow \top}\ \mathsf{trans\_true}$$

$$\frac{\Psi \vdash A \rightsquigarrow G_1 \qquad \Psi, x : A \vdash F \rightsquigarrow G_2}{\Psi \vdash \forall x : A.\, F \rightsquigarrow G_1 \supset \forall x.\, G_2}\ \mathsf{trans\_all} \qquad \frac{\Psi \vdash A \rightsquigarrow G_1 \qquad \Psi, x : A \vdash F \rightsquigarrow G_2}{\Psi \vdash \exists x : A.\, F \rightsquigarrow G_1 \wedge \exists x.\, G_2}\ \mathsf{trans\_ex}$$

$$\frac{\Psi, \underline{x} : (L; \sigma) \vdash (\text{block } L)[\sigma] \rightsquigarrow \Delta; \sigma' \qquad \Psi, \underline{x} : (L; \sigma) \vdash F \rightsquigarrow G}{\Psi \vdash \forall \underline{x} : (L; \sigma).\, F \rightsquigarrow \forall \underline{x} : L.\, (\Delta \supset G)}\ \mathsf{trans\_allP}$$

$$\frac{\Psi, \underline{x} : (L; \sigma) \vdash (\text{block } L)[\sigma] \rightsquigarrow \Delta; \sigma' \qquad \Psi, \underline{x} : (L; \sigma) \vdash F \rightsquigarrow G}{\Psi \vdash \exists \underline{x} : (L; \sigma).\, F \rightsquigarrow \exists \underline{x} : L.\, (\Delta \wedge G)}\ \mathsf{trans\_exP}$$

$$\frac{\Psi \vdash F_1 \rightsquigarrow G_1 \qquad \Psi \vdash F_2 \rightsquigarrow G_2}{\Psi \vdash F_1 \wedge F_2 \rightsquigarrow G_1 \wedge G_2}\ \mathsf{trans\_and} \qquad \frac{\Psi \vdash F_1 \rightsquigarrow G_1 \qquad \Psi \vdash F_2 \rightsquigarrow G_2}{\Psi \vdash F_1 \supset F_2 \rightsquigarrow G_1 \supset G_2}\ \mathsf{trans\_imp}$$

Fig. 6. Translation of Formulas & Non-atomic Types.

using conjunction: we use $\Delta \wedge G'$ as short-cut for

$$\Delta \wedge G = \begin{cases} G & \text{if } \Delta = \cdot \\ \Delta' \wedge (G' \wedge G) & \text{if } \Delta = \Delta', G'. \end{cases}$$

The other rules in Figure 6 are self explanatory. As example, consider the induction hypothesis $\mathbf{ih}_2$ in our type preservation example in Section 2. Its translation results in

$$\Psi \vdash\ \forall t : \text{tp}.\, \forall u : \text{of}\ (E_1'\ V_2)\ \text{t}.\, \exists q : \text{of}\ V\ \text{t}.\, \top$$
$$\rightsquigarrow \forall t.\, (\text{of}\ E_1') \wedge (\text{of}\ V_2) \wedge (\text{of}\ t) \supset (\text{of}\ V) \wedge (\text{of}\ t)$$

The resulting approximation formula encodes the rule that objects of the type family "of" with subobjects V and t can be constructed from an object of type family "of" with subobjects E_1', V_2, for *any* t.

4.1.3 Contexts and blocks.

The final two transformation judgments for contexts and blocks define how to transform an $\mathcal{M}_\omega^+$ context Ψ into an approximation context Δ, and how to transform a block Γ into an approximation context. Note, that Ψ may only include variables of the form $x : A$, $\underline{x} \in (L; \sigma)$, and $\mathbf{x} \in F$, and Γ only parameter declarations of the form $x : A$ because of its location in the block definition. The second judgments also returns a substitution, that appropriately substitutes projections for fixed parameters in Γ. The rules for both judgments are given in Figure 7. Rule ctx_lf, for example, defines how param-

$$\frac{}{\vdash \cdot \rightsquigarrow \cdot}\ \mathsf{ctx\_imp} \qquad \frac{\vdash \Psi \rightsquigarrow \Delta \quad \Psi \vdash F \rightsquigarrow G}{\vdash \Psi, \mathbf{x} \in F \rightsquigarrow \Delta, G}\ \mathsf{ctx\_for}$$

$$\frac{\vdash \Psi \rightsquigarrow \Delta \quad \Psi, \underline{x} : (L;\sigma) \vdash (\text{block } L)[\sigma] \rightsquigarrow \Delta';\sigma'}{\vdash \Psi, \underline{x} : (L;\sigma) \rightsquigarrow \Delta, \underline{x} : L, \Delta'}\ \mathsf{ctx\_block} \qquad \frac{\vdash \Psi \rightsquigarrow \Delta \quad \Psi \vdash A \rightsquigarrow G}{\vdash \Psi, x : A \rightsquigarrow \Delta, G, x}\ \mathsf{ctx\_lf}$$

$$\frac{}{\Psi; \underline{x} : (L;\sigma) \vdash \cdot \rightsquigarrow \cdot; \mathrm{id}}\ \mathsf{block\_empty}$$

$$\frac{\Psi; \underline{x} : (L;\sigma) \vdash \Gamma \rightsquigarrow \Delta;\sigma' \quad \Psi; \underline{x} : (L;\sigma) \vdash A[\sigma'] \rightsquigarrow_a G}{\Psi; \underline{x} : (L;\sigma) \vdash (\Gamma, x : A) \rightsquigarrow (\Delta, G); (\sigma', \pi_x(\underline{x})/x)}\ \mathsf{block\_cons}$$

Fig. 7. Translation of Contexts.

eters $x : A$ declared in Ψ are translated into two declarations in Δ: the name of the parameter x, and the approximation formula that corresponds to A.

This completes the translation of $\mathcal{M}^+_\omega$ contexts and formulas to $\mathcal{P}^+_\omega$. In the next section we present the underlying concept of substitution for variables in $\mathcal{P}^+_\omega$.

4.2 $\mathcal{P}^+_\omega$ Substitutions

Approximation formulas have some non-standard properties. Quantifiers, for example can only be substituted by variable names, and not composite terms. How could one make sense of the approximation formula $(a\ x)$, otherwise? On the other hand there are reasons why one would like to let $\mathcal{P}^+_\omega$-approximation formulas to be be modeled closely to $\mathcal{M}^+_\omega$ proofs. The soundness proof of $\mathcal{P}^+_\omega$ from Section 5, for example, requires exactly this. Lemmas may be applied in an $\mathcal{M}^+_\omega$ proof in several different ways, each time instantiated with different arguments. Consider a situation where the lemma $\forall x : \exp.\,(\mathrm{p}\ x) \supset (\mathrm{q}\ x)$ is applied to "app $E_1\ E_2$" and "lam $(\lambda x.x)$", respectively. The former term contains two free variables, E_1 and E_2, and the latter is closed. Intuitively, one would expect that the corresponding approximation formulas are similar but structurally different from the original formulation of the lemma.

$$(\mathrm{p}\ E_1 \wedge \mathrm{p}\ E_2) \supset (\mathrm{q}\ E_1 \wedge \mathrm{q}\ E_2)$$
$$\top \supset \top$$

Our notion of substitution provides mechanisms to this effect. It is defined with respect to the translation relation of atomic types and objects from Figure 5. For a context Ψ, its translation Δ, and a context of local variables Γ, Figure 8 defines how to apply a non-structure preserving but capture avoiding substitution of a term M (valid in Ψ) into a formula G is written as $G[M/x]$. Substituting block variables $\underline{y}$ for block variables $\underline{x}$ in G on the other hand is standard (written as $G[\underline{y}/\underline{x}]$). We begin now with the discussion of the proof theory for $\mathcal{P}^+_\omega$.

$$G[M/x] = \begin{cases} G' & \text{if } G = (a\ x) \text{ and } \Psi;\Gamma \vdash M \rightsquigarrow_a G' \\ (a\ y) & \text{if } G = (a\ y) \\ \forall y.\,(G'[M/x]) & \text{if } G = \forall y.\,G' \\ \exists y.\,(G'[M/x]) & \text{if } G = \exists y.\,G' \\ \forall \underline{y} : L.\,(G'[M/x]) & \text{if } G = \forall \underline{y} : L.\,G' \\ \exists \underline{y} : L.\,(G'[M/x]) & \text{if } G = \exists \underline{y} : L.\,G' \\ (G_1[M/x] \wedge G_2[M/x]) & \text{if } G = G_1 \wedge G_2 \\ (G_1[M/x] \supset G_2[M/x]) & \text{if } G = G_1 \supset G_2 \\ \top & \text{if } G = \top \end{cases}$$

Fig. 8. Definition of Substitution.

4.3 Proof Theory for $\mathcal{P}^+_\omega$

Applying the translation on some proof state of the meta theorem prover with context Ψ and goal formula F results in a set of approximation formulas Δ and a goal approximation formula G. The proof planning task consists in proving G from Δ. To this end we define a natural deduction calculus for approximation formulas as the proof theory for $\mathcal{P}^+_\omega$. As judgment we write $\Delta \vdash_{\mathsf{nd}} G$ whose meaning is given in form of inference rules in Figure 9. The logic for $\mathcal{P}^+_\omega$ resembles a fragment of intuitionistic first-order logic since it consists of propositional formulas augmented with first-order quantifiers. Its non-standard features include quantification over block variables.

$$\frac{G \in \Delta}{\Delta \vdash_{\mathsf{nd}} G}\ \mathsf{pp\_ax} \qquad \frac{}{\Delta \vdash_{\mathsf{nd}} \top}\ \mathsf{pp\_true} \qquad \frac{\Delta, x \vdash_{\mathsf{nd}} G}{\Delta \vdash_{\mathsf{nd}} \forall x.\,G}\ \mathsf{pp\_allI} \qquad \frac{\Delta \vdash_{\mathsf{nd}} \forall x.\,G}{\Delta \vdash_{\mathsf{nd}} G[M/x]}\ \mathsf{pp\_allE}$$

$$\frac{\Delta, \underline{x} : L \vdash_{\mathsf{nd}} G}{\Delta \vdash_{\mathsf{nd}} \forall \underline{x} : L.\,G}\ \mathsf{pp\_allPI} \qquad \frac{\underline{y} : L \in \Delta \qquad \Delta \vdash_{\mathsf{nd}} \forall \underline{x} : L.\,G}{\Delta \vdash_{\mathsf{nd}} G[\underline{y}/\underline{x}]}\ \mathsf{pp\_allPE}$$

$$\frac{\Delta \vdash_{\mathsf{nd}} G[M/x]}{\Delta \vdash_{\mathsf{nd}} \exists x.\,G}\ \mathsf{pp\_exI} \qquad \frac{\Delta \vdash_{\mathsf{nd}} \exists x.\,G \qquad \Delta, x, G \vdash_{\mathsf{nd}} G'}{\Delta \vdash_{\mathsf{nd}} G'}\ \mathsf{pp\_exE}$$

$$\frac{\underline{y} : L \in \Delta \qquad \Delta \vdash_{\mathsf{nd}} G[\underline{y}/\underline{x}]}{\Delta \vdash_{\mathsf{nd}} \exists \underline{x} : L.\,G}\ \mathsf{pp\_exPI} \qquad \frac{\Delta \vdash_{\mathsf{nd}} \exists \underline{x} : L.\,G \qquad \Delta, \underline{x} : L, G \vdash_{\mathsf{nd}} G'}{\Delta \vdash_{\mathsf{nd}} G'}\ \mathsf{pp\_exPE}$$

$$\frac{\Delta \vdash_{\mathsf{nd}} G_1 \qquad \Delta \vdash_{\mathsf{nd}} G_2}{\Delta \vdash_{\mathsf{nd}} G_1 \wedge G_2}\ \mathsf{pp\_andI} \qquad \frac{\Delta \vdash_{\mathsf{nd}} G_1 \wedge G_2}{\Delta \vdash_{\mathsf{nd}} G_1}\ \mathsf{pp\_andE_1} \qquad \frac{\Delta \vdash_{\mathsf{nd}} G_1 \wedge G_2}{\Delta \vdash_{\mathsf{nd}} G_2}\ \mathsf{pp\_andE_2}$$

$$\frac{\Delta, G_1 \vdash_{\mathsf{nd}} G_2}{\Delta \vdash_{\mathsf{nd}} G_1 \supset G_2}\ \mathsf{pp\_impI} \qquad \frac{\Delta \vdash_{\mathsf{nd}} G_2 \supset G_1 \qquad \Delta \vdash_{\mathsf{nd}} G_2}{\Delta \vdash_{\mathsf{nd}} G_1}\ \mathsf{pp\_impE}$$

Fig. 9. $\mathcal{P}^+_\omega$'s Proof Theory.

In order to illustrate the proof planning process, we apply the $\mathcal{P}^+_\omega$ calculus to find a proof plan for the third case of the case split over D in the type

preservation theorem from Section 2. The abstraction of the proof state after case splitting results in the proof obligation

$$\left.\begin{array}{l} E_1, E_2, E_1', V_2, V, T, D_1, D_2, D_3, P,\ (\text{eval}\ E_1) \wedge (\text{eval}\ E_1'), \\ (\text{eval}\ E_2) \wedge (\text{eval}\ V_2),\ (\text{eval}\ E_1') \wedge (\text{eval}\ V_2) \wedge (\text{eval}\ V),\ (\text{of}\ E_1) \wedge (\text{of}\ E_2) \wedge (\text{of}\ T) \\ \mathbf{ih}_1 \in \forall t.\, (\text{of}\ E_1) \wedge (\text{of}\ t) \supset (\text{of}\ E_1') \wedge (\text{of}\ t) \\ \mathbf{ih}_2 \in \forall t.\, (\text{of}\ E_2) \wedge (\text{of}\ t) \supset (\text{of}\ V_2) \wedge (\text{of}\ t) \\ \mathbf{ih}_3 \in \forall t.\, (\text{of}\ E_1') \wedge (\text{of}\ V_2) \wedge (\text{of}\ t) \supset (\text{of}\ V) \wedge (\text{of}\ t) \end{array}\right\} \Delta$$
$$\vdash_{\mathsf{nd}} (\text{of}\ V) \wedge (\text{of}\ T)$$

To prove $\Delta \vdash_{\mathsf{nd}} (\text{of}\ V) \wedge (\text{of}\ T)$ we first apply the rule pp_impE and obtain the subgoals

(2) $\Delta \vdash_{\mathsf{nd}} (\text{of}\ E_1') \wedge (\text{of}\ V_2) \wedge (\text{of}\ T) \supset (\text{of}\ V) \wedge (\text{of}\ T)$
(3) $\Delta \vdash_{\mathsf{nd}} (\text{of}\ E_1') \wedge (\text{of}\ V_2) \wedge (\text{of}\ T)$

Applying pp_allE on (2) results in

(4) $\Delta \vdash_{\mathsf{nd}} \forall t.\, (\text{of}\ E_1') \wedge (\text{of}\ V_2) \wedge (\text{of}\ t) \supset (\text{of}\ V) \wedge (\text{of}\ t)$

The goal (4) can be immediately discharged by pp_ax as the goal formula occurs in Δ. It remains to prove (3): Applying pp_andI on (3) results in the subgoals

(5) $\Delta \vdash_{\mathsf{nd}} (\text{of}\ E_1')$
(6) $\Delta \vdash_{\mathsf{nd}} (\text{of}\ V_2) \wedge (\text{of}\ T)$

On (5) we apply $\mathsf{pp\_andE}_1$ and obtain

(7) $\Delta \vdash_{\mathsf{nd}} (\text{of}\ E_1') \wedge (\text{of}\ T)$

Applying pp_implE results in

(8) $\Delta \vdash_{\mathsf{nd}} (\text{of}\ E_1) \wedge (\text{of}\ T) \supset (\text{of}\ E_1') \wedge (\text{of}\ T)$
(9) $\Delta \vdash_{\mathsf{nd}} (\text{of}\ E_1) \wedge (\text{of}\ T)$

The proof of (8) is analogously to the proof of (2). From $\mathsf{pp\_andE}_2$ on (9) we obtain $(\text{of}\ E_1) \wedge (\text{of}\ T) \wedge (\text{of}\ E_2)$ which is immediately proved by pp_ax. It then remains to prove (6), whose proof is analogous to the proof of (7). In pseudo notation, one possible representation of the proof plan for $\Delta \vdash_{\mathsf{nd}} (\text{of}\ V) \wedge (\text{of}\ T)$ is therefore

$$\begin{array}{l} \mathsf{pp\_impE}(\ \mathsf{pp\_allE}(\mathsf{pp\_ax}(\forall t.\, (\text{of}\ E_1') \wedge (\text{of}\ V_2) \wedge (\text{of}\ t))), \\ \quad \mathsf{pp\_andI}(\ \mathsf{pp\_andE}_1(\mathsf{pp\_impE}(\ \mathsf{pp\_allE}(\mathsf{pp\_ax}(\forall t.\, (\text{of}\ E_1) \wedge (\text{of}\ t) \supset (\text{of}\ E_1') \wedge (\text{of}\ t))), \\ \qquad \mathsf{pp\_andE}_2(\mathsf{pp\_ax}((\text{of}\ E_1) \wedge (\text{of}\ T) \wedge (\text{of}\ E_2))))), \\ \quad \mathsf{pp\_impE}(\ \mathsf{pp\_allE}(\mathsf{pp\_ax}(\forall t.\, (\text{of}\ E_2) \wedge (\text{of}\ t) \supset (\text{of}\ V_2) \wedge (\text{of}\ t))), \\ \qquad \mathsf{pp\_andE}_2(\mathsf{pp\_ax}((\text{of}\ E_1) \wedge (\text{of}\ E_2) \wedge (\text{of}\ T)))))). \end{array}$$

We are hopeful that the proof theory $\mathcal{P}_\omega^+$ leads to attractive algorithms and implementations which can draw on previous research result in the field of first-order theorem proving. The small proof plan from above illustrates already

how our proof planner can benefit form techniques that handle associativity and commutativity efficiently.

5 Meta Theory

What remains to be discussed is the soundness of $\mathcal{P}^+_\omega$. When a proof state is converted into an approximation context and approximation formula in $\mathcal{P}^+_\omega$ for which no proof plan exist, we can show that the original goal in $\mathcal{M}^+_\omega$ without further splits must remain unprovable. Conversely, the soundness property states that if a formula is provable in $\mathcal{M}^+_\omega$ without applications of the case or recursion rule, we can always construct a derivation of the translated goal in $\mathcal{P}^+_\omega$. We assume implicitly that the entire LF signature Σ is translated into a signature of $\mathcal{P}^+_\omega$ lemmas that can be used freely. The soundness proof is carefully divided into a series of lemmas whose proofs can be found in the accompanying technical report [SA02]. First, we point out that the translation from $\mathcal{M}^+_\omega$ to $\mathcal{P}^+_\omega$ is unique.

Lemma 5.1 (Uniqueness)

(i) *If* $\mathcal{P}_1 :: \Psi \vdash F \rightsquigarrow G$ *and* $\mathcal{P}_2 :: \Psi \vdash F \rightsquigarrow G'$ *then* $G = G'$

(ii) *If* $\mathcal{P}_1 :: \Psi \vdash A \rightsquigarrow G$ *and* $\mathcal{P}_2 :: \Psi \vdash A \rightsquigarrow G'$ *then* $G = G'$

(iii) *If* $\mathcal{P}_1 :: \Psi; \Gamma \vdash A \rightsquigarrow_a G$ *and* $\mathcal{P}_2 :: \Psi; \Gamma \vdash A \rightsquigarrow_a G'$ *then* $G = G'$

(iv) *If* $\mathcal{P}_1 :: \Psi; \Gamma \vdash M \rightsquigarrow_a G$ *and* $\mathcal{P}_2 :: \Psi; \Gamma \vdash M \rightsquigarrow_a G'$ *then* $G = G'$

Proof. By simultaneous induction on the structure of $\mathcal{P}_1, \mathcal{P}_2$, in all four cases. □

Next, we show several substitution lemmas including a lemma that shows that the concept of non-structure preserving substitutions from Section 4.2 is sound. We write $\Delta[M/x]$ for substituting all occurrences of x in Δ in turn.

Lemma 5.2 (Substitution $\mathcal{P}^+_\omega$) *Let* Δ_1 *be an approximation context as the result of translating* Ψ *in which* N *is valid. If* $\mathcal{D} :: \Delta_1, x, \Delta_2 \vdash G$ *then* $\Delta_1, \Delta_2[N/x] \vdash G[N/x]$.

Proof. By induction on $\mathcal{D}$. □

A similar substitution lemma holds for the translation relation. Again, we use a suffix notation $\Psi[N/x]$ to denote the substitution of all x in Ψ by N. Note, that this substitution is the standard substitution common to LF and $\mathcal{M}^+_\omega$ and not the one defined for $\mathcal{P}^+_\omega$ in Section 4.2.

Lemma 5.3 (Substitution Translation) *Let* Ψ_1 *be a context, and* N *valid in* Ψ_1.

(i) *If* $\mathcal{P} :: \Psi_1, x : B, \Psi_2 \vdash F \rightsquigarrow G$
then $\Psi_1, \Psi_2[N/x] \vdash F[N/x] \rightsquigarrow G[N/x]$.

(ii) *If* $\mathcal{P} :: \Psi_1, x : B, \Psi_2 \vdash A \rightsquigarrow G$
then $\Psi_1, \Psi_2[N/x] \vdash A[N/x] \rightsquigarrow G[N/x]$.

(iii) *If* $\mathcal{P} :: \Psi_1, x : B, \Psi_2; \Gamma \vdash A \rightsquigarrow_a G$
then $\Psi_1, \Psi_2[N/x]; \Gamma[N/x] \vdash A[N/x] \rightsquigarrow_a G[N/x]$.

(iv) *If* $\mathcal{P} :: \Psi_1, x : B, \Psi_2; \Gamma \vdash M \rightsquigarrow_a G$
then $\Psi_1, \Psi_2[N/x]; \Gamma[N/x] \vdash M[N/x] \rightsquigarrow_a G[N/x]$.

Proof. By simultaneous induction on the structure of $\mathcal{P}$, in all four cases. □

If an LF type is inhabited by an LF object, we can guarantee that there is a proof plan in $\mathcal{P}^+_\omega$ that corresponds to this object. However, we do not make any claims related to the efficiency of the proof plan. On the contrary, we suspect that the proof plans that are generated by the algorithms contained in the proof of the Filling Lemma 5.4 are in general unnecessarily big. Intuitively, one can see those proof plans as a recipe to construct the filling object directly.

Without loss of generality [HHP93], we can assume that the witness LF object is canonical, which allows us to look only for canonical derivations. The two judgments defining canonical objects are $\Psi \vdash M \Uparrow A$ for canonical forms, and $\Psi \vdash M \downarrow A$ for atomic objects.

Lemma 5.4 (Filling)

(i) *If* $\mathcal{D} :: \Psi \vdash M \Uparrow A$ *and* $\vdash \Psi \rightsquigarrow \Delta$ *then* $\Psi \vdash M \rightsquigarrow G$ *and* $\Delta \vdash_{\mathsf{nd}} G$.

(ii) *If* $\mathcal{D} :: \Psi \vdash M \downarrow A$ *and* $\vdash \Psi \rightsquigarrow \Delta$ *then* $\Psi \vdash M \rightsquigarrow G$ *and* $\Delta \vdash_{\mathsf{nd}} G$.

Proof. By simultaneous induction on the structure of $\mathcal{D}$, in both cases, using Lemma 5.1, 5.2, and 5.3. □

Lemma 5.4 does not include the application of any lemmas, which might be present in Ψ. Those are treated in the proof of the soundness theorem. It asserts that the proof planner will not skip valid proof plans.

Theorem 5.5 (Soundness) *Let* $\mathcal{D} :: \Psi \vdash P \in F$ *where* $\mathcal{D}$ *does neither contain any applications of the case rule nor the recursion rule. Furthermore, let* $\vdash \Psi \rightsquigarrow \Delta$ *and* $\Psi \vdash F \rightsquigarrow G$. *Then* $\Delta \vdash_{\mathsf{nd}} G$.

Proof. By induction on the structure of $\mathcal{D}$ using Lemmas 5.1, 5.2, 5.3, and Lemma 5.4. □

6 Conclusion

We have presented a proof planning technique for $\mathcal{M}^+_\omega$. The meta logic $\mathcal{M}^+_\omega$ formalizes reasoning about deductive systems that are represented in the logical framework. Although it is expressive, elegant, and concise, its case analysis rule is one of the reasons for the bushiness of the search space which renders automated reasoning techniques still difficult. The heuristic approach that has been implemented in the meta theorem prover for $\mathcal{M}^+_\omega$ as part of the Twelf

system, for example, does not scale well. Hardwired specialized heuristics have made theorem proving in this setting feasible to a large extend, but the general problem seems to require more general proof search strategies than are provided by the current implementation.

The present paper proposes a proof planning technique that is based on approximating the information contained in a proof state. Intuitively, it uses dependent type information to study the information flow within a proof. Based on a careful analysis of the proof state, our criterion can in general quickly predict the prospects of a proof state for success. In addition, it is decidable. If there is hope the meta theorem prover is asked to work out the details, if not case analysis over other assumptions or forward lemma application must be considered.

A proof plan is a natural deduction derivation in $\mathcal{P}^+_\omega$ which is decidable. In addition, $\mathcal{P}^+_\omega$ is sound with respect to $\mathcal{M}^+_\omega$. If no proof plan can be found, a proof is guaranteed not to exist.

In future work we plan to implement $\mathcal{P}^+_\omega$ into the Twelf system [PS99]. This implementation will allow us experiment with proof planning in a dependent, higher-order type theoretic setting. In addition, we believe that it will open up research opportunities related to interpreting proof plans operationally, which can be passed to the different components of the theorem proving system. It seems, that the filling operation, for example, could greatly profit from information contained in proof plans, such as which lemmas to apply next and which to never try.

References

[App01] Andrew W. Appel. Foundational proof-carrying code. In *16th Annual IEEE Symposium on Logic in Computer Science (LICS '01)*, pages 247–258, Boston, USA, June 2001.

[BM79] Robert S. Boyer and J. Strother Moore. *A Computational Logic*. ACM monograph series. Academic Press, New York, 1979.

[Bun88] Alan Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction (CADE-9)*, LNCS 310, pages 111–120, Argonne, Illinois, USA, 1988. Springer.

[Coq91] Thierry Coquand. An algorithm for testing conversion in type theory. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 255–279. Cambridge University Press, 1991.

[DFH+93] Gilles Dowek, Amy Felty, Hugo Herbelin, Gérard Huet, Chet Murthy, Catherine Parent, Christine Paulin-Mohring, and Benjamin Werner. The Coq proof assistant user's guide. Rapport Techniques 154, INRIA, Rocquencourt, France, 1993. Version 5.8.

[GW79] Milner A. J. Gordon, M. J. and C. P. Wadsworth. *Edinburgh LCF:A Mechanised Logic of Computation*. LNCS 78. Springer-Verlag, 1979.

[GW92] Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–389, 1992.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[Hut96] Dieter Hutter. Inka - the next generation. In *13th International Conference on Automated Deduction, New Brunswick, USA*, 1996.

[LP92] Zhaohui Luo and Robert Pollack. The LEGO proof development system: A user's manual. Technical Report ECS-LFCS-92-211, University of Edinburgh, May 1992.

[MP91] Spiro Michaylov and Frank Pfenning. Natural semantics and some of its meta-theory in Elf. In L.-H. Eriksson, L. Hallnäs, and P. Schroeder-Heister, editors, *Proceedings of the Second International Workshop on Extensions of Logic Programming*, pages 299–344, Stockholm, Sweden, January 1991. Springer-Verlag LNAI 596.

[Nec97] George C. Necula. Proof-carrying code. In Neil D. Jones, editor, *Conference Record of the 24th Symposium on Principles of Programming Languages (POPL'97)*, pages 106–119, Paris, France, January 1997. ACM Press.

[ORS92] S. Owre, J.M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *Procedings of the 11th International Conference on Automated Deduction (CADE-11)*, pages 748–752, Saratoga Springs, New York, June 1992. Springer Verlag LNAI 607.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer-Verlag LNCS 828, 1994.

[Pfe91] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[Pfe99] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers, 1999. In preparation.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.

[SA02] Carsten Schürmann and Serge Autexier. Towards proof planning for $\mathcal{M}_\omega^+$. Technical Report Yale/DCS/TR1231, Yale University, Computer Science, New Haven, CT, USA, June 2002.

[Sch00] Carsten Schürmann. *Automating the Meta-Theory of Deductive Systems.* PhD thesis, Carnegie Mellon University, 2000. CMU-CS-00-146.

[Sch01a] Carsten Schürmann. Recursion for higher-order encodings. In Laurent Fribourg, editor, *Proceedings of the Conference on Computer Science Logic (CSL 2001)*, pages 585–599, Paris, France, August 2001. Springer Verlag LNCS 2142.

[Sch01b] Carsten Schürmann. A type-theoretic approach to induction with higher-order encodings. In *Proceedings of the Conference on Logic for Programming, Artificial Intelligence and Reasoning(LPAR 2001)*, pages 266–281, Havana, Cuba, 2001. Springer Verlag LNAI 2250.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. MIT Press/Elsevier, 1990.