A Conservative Extension of First-order Logic and its Applications to Theorem Proving

David Basin and Seán Matthews

MPI–I–93–235 December 1993

Author's Address

Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany.
`{basin,sean}@mpi-sb.mpg.de`

Publication Notes

This paper will appear in the proceedings of the thirteenth conference on theoretical computer science and foundations of software technology, Bombay, December 1993, published in Springer Lecture Notes in Computer Science.

Abstract

We define a weak second-order extension of first-order logic. We prove a second-order cut elimination theorem for this logic and use this to prove a conservativity and a realisability result. We give applications to formal program development and theorem proving, in particular, in modeling techniques in formal metatheory.

Keywords

First-order logic. Higher order logic. Theorem proving.

§ 1 Introduction

A basic problem for the designer of a system for machine checked mathematics is how to allow a user to extend it safely. To this end all sorts of machineries have been imagined and implemented; what they all share (hopefully) is that they do not introduce logical inconsistencies that allow the user to prove false sentences.

Perhaps the best known ways to extend a theorem proving system are lemmas and tactics, which were systematically explored in [7]; also common are metatheoretic reasoning and reflection [14]. Different approaches will, naturally, be successful in different situations; for instance a lemma mechanism (i.e. a library facility that allows a user to take advantage of a collection of previously proven theorems) is the simplest, but it only works well in theories (mostly strong 'foundational' ones, such as set theory or type theory). The problem with a lemma facility in, for example, first-order logic, is that any particular theorem is too limited a statement to be generally usable. For instance a simple truth about first-order logic is that, for any formula A,

$$\vdash \forall x A \to \neg\exists x \neg A. \tag{1}$$

But this is not, itself, a statement of first-order logic. Certainly, for any particular instance of A the formula $\forall x A \to \neg\exists x \neg A$ is a theorem, but particular instances are not very useful; a particular instance is unlikely to come round again, even though exactly the same pattern (in this case even with exactly the same justifying proof steps) probably will. The problem is that (1) is *schematic*; i.e. A varies over formulae, and in a simple first-order language it is not possible to take the universal closure of such a statement, which could then be proven once, to be instantiated as necessary. Some theories are more flexible in the way that they allow quantifiers to be used though, and then the problem disappears; e.g. in a type theory, where we can quantify over formulae, we are able to write (and prove) the universal closure of (1) as

$$\forall A(\forall x A(x) \to \neg\exists x \neg A(x)). \tag{2}$$

Another possibility is available in set theory, which, though it is first-order, and so does not allow quantification over formulae, has comprehension axioms that produce, for any A a c_A such that $A(x) \leftrightarrow x \in c_A$; and then we can get the same effect with the provable statement

$$\forall c(\forall x(x \in c) \to \neg\exists x \neg(x \in c)).$$

The problem with moving to theories like type theory or set theory is that they introduce new complexities that may be inconvenient in other ways. There are many useful theorem proving tools that work well in simple theories, but become very inefficient, or even do not work at all, in these much more powerful theories; and for most purposes, such theories provide enormously more than we need — a lot of the time we use them just so that we can make use of such simple tricks.

Since (1) cannot be expressed in ordinary first-order logic, we need some other approach if the user is to be able to make use of it in a theorem prover, and this is one of the reasons for tactic languages.[1] Even if we cannot write some general statement that we can see to be true, if we have a tactic language we can still express its proof: we can write a program that automatically builds the proof whenever it is needed, so that we do not have to go to the slow and error-prone bother of typing it in by hand each time. This is obviously going to be safe because it actually goes to the effort of building the proof each time it is needed; it does not modify the system, it only speeds up, by automation, the interface; i.e. the resulting proofs are, rule for rule, identical with what would have been produced manually.

The other way to extend a theorem proving system is much more powerful, but also much more difficult. The theory of a theorem prover is a mathematical object itself, so we can formalise it too, and then do formal meta-theory, which we can use to verify that extensions we make to the system are safe. The problems with this approach are that metamathematics is unusually awkward to formalise in a way that is practical (see [2, 11]) and that we now have to maintain two formal systems: the original theory, and its metatheory. We can fix this second problem by attempting something even more ambitious: if we have formal theory strong enough to encode and reason about other formal theories, then we can use it to formalise itself and use it as its own meta-theory. This is a powerful way to extend a system [3, 9, 14] but requires that complex and subtle metamathematics be formalised inside the theorem prover.

In this paper we present a different way to extend a theorem proving system, that allows us to state and prove formulae like (2). We describe a second-order extension of a first-order logic where 'schematic' formulae such as (2) can be stated and proved, without introducing the complexity of a full type theory. The logic we present has several useful properties. First, it is expressive enough to state and prove theorems using the second-order syntax. We do this by allowing quantification over abstracted first-order formulae (which we can think of as formulae containing 'holes'). This gives us schematic facilities similar to what are available in (the second-order fragment of) logical frameworks such as *Isabelle*. With these new facilities a theory for, e.g., first-order logic, can be extended to provide directly the same kinds of derived rules that *Isabelle* allows in the metatheory. Secondly, it is a conservative extension: no new formulae in the language of the original logic become provable. This conservativity result is shown constructively, using a elimination theorem for those cuts that introduce second-order features. (This is not a property of most stronger logics: for example ordinary second-order logic[13] is certainly not conservative over first-order logic.)

The rest of our paper is organized as follows: in § 2 we define our notation and give the background that a reader needs, then we introduce our basic theory LK_2 and prove a cut elimination theorem showing that it has the properties we need; in § 3 we show that our theory is adequate for the purposes we have discussed here — in particular, meta-theory and 'schematic proofs'; finally, in § 4, we draw conclusions.

[1]Of course there are others too: for instance to provide heuristic procedures, which cannot properly be formalised as lemmas.

§ 2 The logic LK_2, and a cut elimination theorem

In this section we describe a second-order extension to the language $\mathcal{L}_{LK}$ which we call $\mathcal{L}_{LK_2}$ and a second-order extension to the logic LK which we call LK_2. Then we prove a restricted cut elimination theorem for LK_2. First though, we give some preliminary definitions.

§ 2.1 Preliminary definitions and conventions To save space we take our definitions, where possible, from [5].

Definition 1 (LK) *Take the sequent calculus presentation of intuitionistic logic in* [5, Ch. 5] *as LJ; then LK is LJ extended with all sequents of the form $\Gamma \vdash A \vee \neg A$; we refer to sequents of this form as* ex-mid*. $\mathcal{L}_{LK}$ is the language of first-order logic associated with LK. We indicate substitution of a free variable x by a term t in an object A by $A[t/x]$.*

We make use of a modified form of the cut elimination proof for LK, and the reader should be familiar with it (again, see [5]); note that the proof is by induction on the structure of derivations, using the lexicographic ordering defined on the pair $\langle R, S \rangle$, where R is the number of connectives in the cut formula at the root of an (otherwise cut-free) proof, and S is the number of steps in the proof.

Now we are able to define the logic LK_2.

Definition 2 ($\mathcal{L}_{LK_2}$) *The language $\mathcal{L}_{LK_2}$ is the smallest extension of the language $\mathcal{L}_{LK}$ closed under the following:*

1. *We define a new class of predicate variables, α, of all arities, where if α has arity n, then $\alpha(t_1, \ldots, t_n)$ is in $\mathcal{L}_{LK_2}$.*

2. *If A is in $\mathcal{L}_{LK_2}$, and α a predicate variable, then $\forall_2 \alpha A$ and $\exists_2 \alpha A$ are in $\mathcal{L}_{LK_2}$.*

3. *If A is in $\mathcal{L}_{LK_2}$* not containing $\forall_2$ or $\exists_2$, *then $\lambda x_1, \ldots, x_n A$ is a predicate of $\mathcal{L}_{LK_2}$ of arity n, and called an abstracted formula.*

We call $\forall_2$ and $\exists_2$ second-order quantifiers. Note that we have to extend our notion of substitution to allow replacement of n-ary predicate variables by n-ary predicates: if an abstracted formula, say $\lambda x_1 \ldots \lambda x_n A$, is substituted for a predicate variable α in a formula like $\alpha(t_1, \ldots, t_n)$, then the result, after appropriate alpha-conversion, is $A[t_1/x_1, \ldots, t_n/x_n]$.

Definition 3 (LK_2) *The logic LK_2 has the language $\mathcal{L}_{LK_2}$ and its rules include those for the logic LK (extended for $\mathcal{L}_{LK_2}$); extra rules are also added for the second-order quantifiers $\forall_2$ and $\exists_2$, as follows:*

$$\frac{\Gamma, A[p/\alpha] \vdash B}{\Gamma, \forall_2 \alpha A \vdash B}\ \forall_2\text{-}l \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall_2 \alpha A}\ \forall_2\text{-}r$$
$$\frac{\Gamma, A \vdash B}{\Gamma, \exists_2 \alpha A \vdash B}\ \exists_2\text{-}l \qquad \frac{\Gamma \vdash A[p/\alpha]}{\Gamma \vdash \exists_2 \alpha A}\ \exists_2\text{-}r \tag{3}$$

Finally, an extra side condition is placed on the first-order rule $\exists$-l: not only should the quantified variable not appear free in the other formulae in the sequent, but there should be no occurrences above it in the proof hidden by an application of $\exists_2$-r. We also place a restriction on ex-mid*, that we do not allow instances containing second-order quantifiers.*

In $\forall_2$-l and $\exists_2$-r the arity of α and p should be the same; also notice that the syntax (definition 2, part 3) places the restriction on p that it be an abstraction of a formula that does not contain any second-order quantifiers.

We now distinguish between different cuts in LK_2.

Definition 4 (second-order cuts) *In the rule*

$$\dfrac{\Gamma\vdash A \quad \Delta, A\vdash B}{\Gamma,\Delta\vdash B}\textit{cut}$$

A is called the cut formula. If A contains either $\forall_2$ or $\exists_2$, then the cut is said to be second-order.

We can now state, and prove, the most important fact about LK_2:

Theorem 5 *If a sequent $\Gamma\vdash A$ is provable in LK_2 then it is provable without using second-order cuts.*

Proof: By induction on derivations, and similar to ordinary cut elimination, except that we treat first-order cuts like any other rule. We use a lexicographic ordering on the triple $\langle Q, R, S\rangle$ where R and S are, as in the usual proof, the complexity of the cut formula, and the number of steps in the derivation, and Q is the number of second-order quantifiers in the cut formula.

Since most of the cases for the induction are the same as for ordinary cut elimination, we will look only at some of the new ones. Consider the case for $\exists_2$; here we have a transformation

$$\dfrac{\dfrac{\begin{array}{c}\vdots\,\Pi_1\\ \Gamma\vdash A[p/\alpha]\end{array}}{\Gamma\vdash\exists_2\alpha A}\quad \dfrac{\begin{array}{c}\vdots\,\Pi_2\\ A,\Delta\vdash B\end{array}}{\Gamma',\exists_2\alpha A\vdash B}}{\Gamma,\Gamma'\vdash B}\text{cut} \quad\rightsquigarrow\quad \dfrac{\begin{array}{c}\vdots\,\Pi_1\\ \Gamma\vdash A[p/\alpha]\end{array}\quad \begin{array}{c}\vdots\,\Pi_2[p/\alpha]\\ \Gamma, A[p/\alpha]\vdash B\end{array}}{\Gamma,\Gamma'\vdash C}\text{cut}$$

where Π_1 and Π_2 are derivations. Notice that $A[p/\alpha]$ does not necessarily have a smaller R measure, since p is an arbitrary formula, but it does have a smaller Q measure, since a second-order quantifier has been removed and, by the restrictions we have placed on it, p cannot have introduced any new ones. By the hypothesis it is now possible to eliminate all second-order cuts in the derivation. The case for $\forall_2$ is very similar. The other extra case is when one of the sub-derivations of a second-order cut is a first-order cut. Then the restructuring needed is

$$\dfrac{\dfrac{\begin{matrix}\vdots\,\Pi_1\\ \Gamma\vdash A\end{matrix}\quad\begin{matrix}\vdots\,\Pi_2\\ \Gamma',A\vdash B\end{matrix}}{\Gamma,\Gamma'\vdash B}\,\mathsf{cut}_1\qquad\begin{matrix}\vdots\,\Pi_3\\ \Gamma'',B\vdash C\end{matrix}}{\Gamma,\Gamma',\Gamma''\vdash C}\,\mathsf{cut}_2\quad\rightsquigarrow\quad\dfrac{\begin{matrix}\vdots\,\Pi_1\\ \Gamma\vdash A\end{matrix}\qquad\dfrac{\begin{matrix}\vdots\,\Pi_2\\ \Gamma',A\vdash B\end{matrix}\quad\begin{matrix}\vdots\,\Pi_3\\ \Gamma'',B\vdash C\end{matrix}}{\Gamma',\Gamma'',A\vdash C}\,\mathsf{cut}_2}{\Gamma,\Gamma',\Gamma''\vdash C}\,\mathsf{cut}_1$$

and, since the S measure of the subderivation ending with the second-order cut is now smaller, it can, by appeal to the hypothesis, be restructured to be cut free; and, if the cut is on the right hand side, the transformation is essentially the same. And the theorem follows.

Readers familiar with the proof of cut elimination will rightly suspect that (if we drop the law of the excluded middle, or rearrange our calculus into the standard classical form) it is possible to eliminate all the cuts from a proof in LK_2. However we are interested only in removing second-order cuts, and we want to make sure this result holds if we extend LK_2 to theories that do not allow elimination of first-order cuts. Further (and ironically), we need excluded middle in order to get the realisability result below in corollary 8.

Remark 6 *Notice that any application of any of the rules in (3) is recorded in the root sequent of a proof, or it is eliminated by a cut somewhere below in the derivation. Thus, in a proof free of second-order cuts we can tell whether or not any of the rules in (3) have been used simply by examining the formulae in the root sequent.*

As an important corollary of theorem 5 we have the following:

Corollary 7 *The logic LK_2 is conservative with respect to LK.*

Proof: If $\Gamma \vdash A$ is a sequent in the language of $\mathcal{L}_{LK}$ provable in LK_2 then, by theorem 5, it is provable without second-order cuts. None of the rules of (3) can have been used in this proof, since any such application would result in a second-order quantifier appearing in the final sequent (since there are no second-order cuts, the only alternative, that second-order quantifiers appear in cut formulae that do not appear in the final sequent, is, by remark 6, impossible). The proof may, however, contain second-order variables; for each of these substitute in any abstracted formula of the right arity, not containing second-order variables. The result is a proof of the sequent in LK.

§ 2.2 Realisability

A second corollary is that if a theorem begins with a second-order existential quantifier, we can ecover a witness for that quantifier:

Corollary 8 *If a sequent $\vdash \exists_2\alpha A$ is provable in LK_2 then it is possible to find an abstraction p such that $\vdash A[p/\alpha]$ is provable in LK_2.*

Proof: We prove a generalisation of this by induction on the depth of derivations, for sequents of the form $\Gamma \vdash \exists_2 \alpha A$, where Γ contains no second-order quantifiers.

If $\Gamma \vdash \exists_2 \alpha A$ is provable in LK_2 then, by theorem 5, it is provable without second-order cuts, and the rule above it must be either $\exists_2$-r or a first-order cut, or one of the first-order left rules. In the first case, the derivation is

$$\dfrac{\begin{matrix}\vdots \\ \Gamma \vdash A[p/\alpha]\end{matrix}}{\Gamma \vdash \exists_2 \alpha A}\ \exists_2\text{-r}$$

so p is the predicate we are looking for. In the case of a first-order cut, the we can restructure the derivation as:

$$\dfrac{\begin{matrix}\vdots \\ \Gamma' \vdash B\end{matrix} \qquad \begin{matrix}\vdots \\ \Gamma'' \vdash \exists_2 \alpha A\end{matrix}}{\Gamma', \Gamma'' \vdash \exists_2 \alpha A}\ \mathsf{cut} \quad \rightsquigarrow \quad \dfrac{\begin{matrix}\vdots \\ \Gamma' \vdash B\end{matrix} \qquad \begin{matrix}\vdots \\ \Gamma'', B \vdash A[p/\alpha]\end{matrix}}{\Gamma', \Gamma'' \vdash A[p/\alpha]}\ \mathsf{cut}$$

where $\Gamma = \Gamma', \Gamma''$. This follows since, by the induction hypothesis, we can derive a witness p for the righthand subgoal. Similarly for all the other left hand rules, except for the rule for disjunction (the only other problem might be with the rule for $\exists$-r, but the side condition on the rule ensures that this problem is avoided.[2] There we have $\Gamma = \Gamma', \Gamma'', B \vee C$ and the derivation is

$$\dfrac{\begin{matrix}\vdots \\ \Gamma', B \vdash \exists_2 \alpha A\end{matrix} \qquad \begin{matrix}\vdots \\ \Gamma'', C \vdash \exists_2 \alpha A.\end{matrix}}{\Gamma', \Gamma'', B \vee C \vdash \exists_2 \alpha A}$$

By the induction hypothesis we can find provable sequents $\Gamma', B \vdash A[p/\alpha]$ and $\Gamma'', C \vdash A[q/\alpha]$ and, if we take (abusing notation slightly) $r \equiv (B \wedge p) \vee (\neg B \wedge q)$ to be the obvious formula abstraction, from these we can derive $\Gamma \vdash A[r/\alpha]$ using excluded middle ($B \vee \neg B$). Note that r is uniformly equivalent in each case to either p or q but never both; i.e. if B is true, then q is 'switched off', no matter what value C has. Here we can see the intuition behind the restriction on excluded middle to formulae free of second-order formulae: in a first-order classical theory we are unable to realise the existentials because we are not able to resolve disjunctions introduced by excluded middle. Here, however, we are trying to realise second-order existential variables, and we can simply factor instances of excluded middle into the formula we are trying to realise. However, this would not be possible if instances of excluded middle could contain second-order quantifiers.

[2]This side condition is in fact almost identical with what the behaviour of *Isabelle* turns out to be in a similar circumstance.

Remark 9 *The design of LK_2 is pragmatic, and with this in mind, the way to extend it to other, more powerful theories, sould be obvious. Consider, for instance, a theory of arithmetic; then cut elimination is not available in the constructive, finitist manner described above, if it is available at all. But we can extend LK_2 to such a theory, while ensuring that the properties above still hold.*

The problem divides naturally into two parts. We extend LK to arithmetic (AR) by first adding, as new basic rules or axioms all sequents of the form $\Gamma \vdash A$, where A is a theorem of elementary arithmetic, and then adding a new rule

$$\frac{\Gamma \vdash A[0/x] \quad \Gamma, A \vdash A[s(x)/x]}{\Gamma \vdash \forall x A}\ \textit{ind.}$$

Then we can convert AR into AR_2 in the same way as we converted LK into LK_2; however we do not carry over ind *to AR_2 in the same simple way: we add the side condition that A* does not contain any second-order quantifiers. *It is this side condition that ensures that if we apply the cut elimination procedure above, we shall never encounter a second-order cut where the cut formula is produced by an induction, and so do not have to try to eliminate such cuts. And thus theorem 5 still holds for AR_2. With this knowledge we can check that the corollaries also hold.*

§ 3 Applications

§ 3.1 Metatheory: schematic theorems Recall the motivation given in the introduction: to provide an extension of a logic where formulae like (1) can be derived as lemmas, instead of each instance having to be proven separately. Clearly we can do this here: since we have a theory in which we can prove the second-order universal closure (2). And we can use the resulting lemma to produce first-order theorems as necessary; i.e. we can instantiate the $\forall_2$ with any first-order abstracted formula to get any instance we need, and we know from the conservativity results that such a first-order theorem really is a theorem. (Note that our conservativity proof actually gives us more than that: because it is constructive we can remove all second-order traces from the final proof.)

For example, induction is typically not finitely axiomatisable in a first-order theory, and has to be provided by a class of axioms defined by a schema. Unfortunately the given induction schema is often not the most convenient for a particular proof: a more natural proof is often possible with some other, more specialised induction principle. In arithmetic, for instance, the standard induction is structural induction on the successor function, but, in practice, course-of-values induction, i.e. that for any formula A

$$\forall x(\forall y(y < x \to A[y/x]) \to A) \to \forall x A,$$

is often easier to use. But the general form of this is not a theorem of, say, AR, even though there is a simple, well known, meta-theoretic proof of the schema, because A ranges over formulae, which are not quantifiable over in the theory. And so each instance has to be proven as needed. But in the extension we propose, we can instead show simply:

Proposition 10 *In the theory AR_2:*

$$\vdash \forall_2 \alpha(\forall x(\forall y(y < x \to \alpha(y)) \to \alpha(x)) \to \forall x \alpha(x)).$$

Proof: Standard. Assuming an arbitrary α and the antecedent of $\to$, show the consequent: we generalize, and prove

$$\forall x(\forall y(y < x \to \alpha(y)) \to \alpha(x))$$

by structural induction. This is the standard proof and is clearly formalizable in AR_2, since the induction formula contains no second-order quantifiers.

§ **3.2 Metatheory: existentially schematic theorems** We have shown that we can formalize schematic theorems by treating metavariables in a theorem as second-order variables and proving the universal second-order closure. But there are other applications of metavariables, where we do not want to show that the formula is provable for all instantiations, but rather that there is *some* instantiation for which it is provable. For example, in [1] we show how what we call 'schematic proofs' can be used for program synthesis, and, as as worked example, show how, in *Isabelle*, we can start with a schematic formula

$$\forall x \forall y \forall z(x + y = z \leftrightarrow \alpha(x, y, z)), \tag{4}$$

and gradually refine α to be a recursive predicate (a logic program for addition) simply by applying the ordinary proof rules of the system, such as induction, with unification gradually filling out α. In effect we have, implicitly, existentially quantified the metavariables, and then, in the process of the proof, constructed witnesses for them. Here we might try to make this implicit quantification explicit by quantifying α existentially (with $\exists_2$) in the theory. But we have to be sure that this will work. In fact we need the following properties:

1. That we can recover valid instantiations of such $\exists_2$ quantified variables,

2. That we can recover valid proofs in the original logic (i.e. proofs that do not use any of the second-order features we have added) supporting these instantiations.

We show that LK_2 has these properties. First, if we represent a formula with schematic variables $\alpha_1 \dots \alpha_n$ as a formula where the α_i are predicate variables quantified using $\exists_2$, then, by corollary 8 above, if we have a proof in a theory such as AR_2 that the latter is a theorem, then we can recover witnesses for the α_i. In fact, we do better than just being able to recover a first-order theorem, since there is no reason why there cannot be $\forall_2$ quantifiers inside the scope of the $\exists_2$, in which case, after realising the α_i, the result is a theorem that is, itself, schematic.

The second property also follows, from the existence of effective witnesses and conservativity: given a proof of a ground first-order formula A, then, *irrespective of* *how the proof makes use of second-order facilities*, by corollary 7 and the fact that its proof is constructive, we can recover a proof of this formula in the original theory.

So we can state (4) in AR_2 as the formula

$$\exists_2 \alpha (\forall x \forall y \forall z (x + y = z \leftrightarrow \alpha(x, y, z))) \tag{5}$$

and prove it, knowing that we can recover a witness for α when we are finished.

Obviously it is not possible here to use the schematic lemmata we have discussed earlier, since we now have occurrences of $\exists_2$ in the theorem itself, rather than just in schematic lemmas in the library. But there are other sorts of lemmata that we can state and prove, and then use here. For instance, in [1] the idea was to refine the α in (4) as necessary for the derivation. In particular, an application of induction was used to refine α to a recursive predicate. We can state and prove a lemma equivalent to that refinement as follows[3]

$$\begin{aligned}&\forall_2 \gamma (\exists_2 \beta \forall y (\gamma(0, y) \leftrightarrow \beta(y)) \rightarrow \\ &\qquad \exists_2 \sigma \forall x \forall y (\gamma(s(x), y) \leftrightarrow \exists y' (\gamma(x, y') \wedge \sigma(s(x), y', y))) \rightarrow \\ &\qquad \exists_2 \alpha (\gamma(x, y) \leftrightarrow \alpha(x, y)))\end{aligned} \tag{6}$$

where γ is a specification, β and σ are the base and step cases, and α will then be a recursive predicate equivalent to γ. Then refining (5) by back chaining through (6) has a similar effect to applying induction to (4).

§ 4 Related and further work, conclusions

We have proposed here a method for extending a theory with second-order quantifiers. This extension provides approximately the same facilities as would be available with predicate reasoning at the meta-level, i.e. simple derived rules and schematic proof development. It can also be seen as a way of importing some of the often claimed advantages of higher order logics (and functional programming languages) into a conservative extension of a first-order theory. We believe this is useful for many purposes, and avoids the complexities that meta-level reasoning proper involves. Similar facilities can be supplied with a logical framework style system such as the LF of Harper, Honsell and Plotkin [8]. The approaches have different applications, of course; the approach here is much simpler to implement, for any particular logic, than a complete type-theoretic framework, and will probably be faster, but then it lacks the universality that an LF style system boasts.

Other similar work on adding (the effect of) our *universal* second-order quantifiers to a theory is that of Boyer and Moore [4], who effectively exploit the theorem on constants to get theorems schematic over classes of functions in a purely first-order setting. Also, Goguen [6] discusses parametrised algebraic specifications, and adding universal quantifiers over functions to equational theories as a way to supply the equivalent of higher order functions in a first-order setting. Interestingly, Goguen gives

[3]We gloss over technical details (such as the problem of defining new recursive predicates) in order to give a *feel* for the idea.

a semantic proof of the conservativity of his approach, and argues that this is better; we are sympathetic to the argument for semantic rather than proof theoretic proofs in general, but we think that the proof-theoretic methods here have the advantage that they not only prove conservativity, but allow all syntactic traces of the extra machinery to be removed; also we are able to treat existential quantifiers, which neither Boyer and Moore nor Goguen discuss.

On the theoretical side, our theory LK_2, and the cut elimination theorem we prove for it, resemble the ramified second-order logic and associated normalisation results Prawitz discusses [12]; the major differences are the modifications we make to get realisability and the embedding of other theories inside it. Also, we use sequent calculus, were he uses natural deduction.

There are many areas of possible further work. For instance, the system as described here is a slightly *ad hoc* modification of intuitionistic sequent calculus, and it would be good to get a more elegant formulation. Also, our realisability result only applies to classical theories; oddly, it seems much more difficult to get a 'nice' realisability result in a constructive logic, but it would be very useful to have. However, even if we sacrifice easy realisability by throwing away the excluded middle rule, the fragment for the universal quantifier certainly still is very useful for the sort of applications that Boyer and Moore, and Goguen describe.

Acknowledgments

We thank Alan Smaill and Nax Mendler for their careful reading of drafts, which uncovered various bugs. Paul Taylor's code for setting proofs was used. The research of the first author was in part supported by the German Ministry for Research and Technology under grant ITS 9102.

Responsibility for the contents of this paper lies with the authors.

References

[1] D. Basin, A. Bundy, I. Kraan, and S. Matthews. A framework for program development based on schematic proof. In *Proc. Seventh International Workshop on Software Specification and Design*, 1993. Also available as MPI-93-231.

[2] D. Basin and R. Constable. Metalogical frameworks. In [10].

[3] R. Boyer and J. S. Moore. Metafunctions: Proving them correct and using them efficiently as new proof procedures. In R. Boyer and J. S. Moore, editors, *The Correctness Problem in Computer Science.* Academic Press, 1981.

[4] —— Functional instantiation in first-order logic. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation.* Academic Press, 1991.

[5] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge University Press, 1989.

[6] J. Goguen. Higher-order functions considered unnecessary for higher-order programming. In D. A. Turner, editor, *Research Topics in Functional Programming*. Addison-Wesley, 1990.

[7] M. J. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, LNCS 78, 1979. Springer.

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

[9] D. J. Howe. Computational metatheory in Nuprl. In *Proc. CADE-9*, LNCS 310, 1988. Springer.

[10] G. Huet and G. Plotkin, editors. *Logical Environments*. Cambridge University Press, 1993.

[11] S. Matthews, A. Smaill, and D. Basin. Experience with FS_0 as a framework theory. In [10].

[12] D. Prawitz. *Natural Deduction; A proof theoretic study*, Almqvist and Wiksell, 1965.

[13] G. Takeuti. On a generalized logical calculus. *Japanese Journal of Mathematics*, 23:39–96, 1953.

[14] R. Weyhrauch. Prolegomena to a theory of mechanised formal reasoning. *Artificial Intelligence*, 13:133–170, 1980.