The HOL/NuPRL Proof Translator
A Practical Approach to Formal Interoperability*

Pavel Naumov[1] Mark-Oliver Stehr[2] José Meseguer[2]

[1] Pennsylvania State University,
Middletown, PA 17057, USA, naumov@psu.edu

[2] SRI International,
Menlo Park, CA 94025, USA, {stehr,meseguer}@csl.sri.com

Abstract. We have developed a proof translator from HOL into a classical extension of NuPRL which is based on two lines of previous work. First, it draws on earlier work by Doug Howe, who developed a translator of theorems from HOL into a classical extension of NuPRL which is justified by a hybrid set-theoretic/computational semantics. Second, we rely on our own previous work, which investigates this mapping from a proof-theoretic viewpoint and gives a constructive meta-logical proof of its soundness. In this paper the logical foundations of the embedding of HOL into this classical extension of NuPRL as well as technical aspects of the proof translator implementation are discussed.

1 Introduction

During the last couple of decades we have witnessed an appearance of several dominant theorem proving systems that have attracted considerable attention and have gradually found their way into applications. Simultaneously, the same process has fragmented the research community and has made work and results of any one group almost useless for the rest. The main reason for this fragmentation is the fact that a majority of modern theorem provers have been designed as stand-alone products that can not share formal results with other systems. Partially this can be explained by the fact that different systems are based on quite different logical foundations, which makes any translation a non-trivial mathematical problem. But probably even more importantly, the designers have seldom seen the need for such compatibility. Their goal was to explore new approaches, not to form standards. The need for compatibility appeared when the leading systems accumulated substantial libraries of formal theories produced by joint work of many users. Reproducing such theories in systems where they are not available would be a very tedious and time-consuming task. At the same time, those other systems have unique and interesting features that make them attractive for some users. As a result, the need for proof translation tools has become apparent.

* The first author was supported by DARPA grant F30602-98-2-0198 on the initial stage of this work that was done at Cornell University, Ithaca, NY. We furthermore gratefully acknowledge support for the work conducted at SRI by DARPA and NASA (Contract NAS2-98073), by Office of Naval Research (Contract N00014-96-C-0114), by NSF Grant (CCR-9633363), and by a DAAD grant in the scope of HSP-III.

Two Kinds of Translators. A translation of formal results from one system into another can be carried out at two different levels. First, one can translate *statements* of proven theorems from a source system into a target system. Secondly, one can translate the *proofs* themselves.

The first approach is obviously easier to implement, but it forces the target system to trust the correctness of the source system. In addition, since systems are usually based on different logical foundations, the user has to rely completely on an (informal) meta-logical proof of the fact that the translator is based on a logically sound mapping. Besides, one can raise doubts about the validity of the code of the translator that can have bugs or, for technical reasons, may slightly deviate from the mapping used in the soundness proof. All these problems can be avoided with the second approach, but the need to translate proofs makes the implementation much harder.

Ideally, there is a close connection between the soundness of the mapping between the two formal systems and proof translation: If $\mathcal{L}$ and $\mathcal{L}'$ are the source and target logics, respectively, and α is the mapping of $\mathcal{L}$ into $\mathcal{L}'$ then *soundness*[3] is the property that $\Gamma \vdash_{\mathcal{L}} P$ implies $\alpha(\Gamma) \vdash_{\mathcal{L}'} \alpha(P)$ for any set Γ of axioms and any formula P. Hence, a proof of soundness would demonstrate that for each proof of P from Γ in $\mathcal{L}$ there is a corresponding proof of $\alpha(P)$ from $\alpha(\Gamma)$ in $\mathcal{L}'$. Furthermore, if the soundness proof is conducted in a constructive way, it implicitly contains a description of the translation algorithm that we need. In fact, in our case the proof translator from HOL into NuPRL is an algorithm, which is essentially extracted from a mathematical proof of soundness.

Obviously, soundness of the underlying mapping is the main theoretical requirement for the correctness of the proof translator. Notice, however, that soundness can always be achieved by extending the target system by additional axioms and inference rules, as it is often necessary in practice. Of course, such an extension could make the target system inconsistent, which is why the soundness proof is meaningful only in the presence of a consistency proof of the target system. Typically, such a consistency proof is done using an abstraction of the target system, e.g. by construction of a semantic model. In a more general setting, where we work relative to arbitrary theories, a suitably general *model preservation* property[4] property is that each model M of $\mathcal{L}$ can be obtained from a model M' of $\mathcal{L}'$ by a mapping β such that $\beta(M') \models_{\mathcal{L}} P$ iff $M' \models_{\mathcal{L}'} \alpha(P)$. Although a proof of model preservation is generally important for the translation mapping to be semantically meaningful, it is of little use for the implementation of the proof translator itself which typically relies on proof-theoretic constructions.

Previous Results. Some early work on translating formal results between theorem provers was conducted by Doug Howe and Amy Felty. In [12] Howe defined a mapping from HOL [9] into a classical extension of NuPRL [3] and gave a seman-

[3] Soundness is the central property of a map of entailments systems, which represent the notion of derivability in general logics [14].

[4] This property is usually expressed in the context of a map of institutions [8], which constitute the model-theoretic component of general logics [14].

tic justification for the admissibility of this translation, essentially by constructing a hybrid set-theoretic/computational semantics for the classical extension of NuPRL[5], where HOL models appear as submodels of somewhat richer NuPRL models [10]. Later, Howe developed a translator [11] based on this mapping, which translates *statements* of HOL theorems. Since Howe gave only a semantic justification, extending it to a proof translation is a non-trivial task. The practical usefulness of Howe's HOL/NuPRL connection has been demonstrated in [7], where his translator is used to produce HOL/NuPRL hybrid proofs. In [17] the first author applied a similar approach to built a translator from Isabelle's higher-order logic [19] into NuPRL. This work also sketched a soundness proof, which was so straightforward that in [18] he has been able to formalize it in NuPRL. A corresponding soundness proof can also be given for Howe's original mapping from HOL into NuPRL. In fact, we gave a detailed constructive proof in [22] using general logics terminology [14]. In this paper we present a proof translator from HOL to a classical extension of NuPRL which is based on this proof. Comparison to other closely related works is done in the end of the paper.

Why HOL and NuPRL? There are at least three factors to consider when selecting the source and the target systems for a proof translator to be useful in practice: (1) The source logic should be supported by a well-established theorem proving system that has accumulated a substantial amount of non-trivial results. (2) The target logic should be sufficiently expressive to serve as a target for the translation, so that the translation is natural in the sense that, say, results about standard data types in the source logic can be interpreted as results about the corresponding data types in the target logic. (3) The target logic should be equipped with a semantics that allows us to establish a close relationship to the semantics of the source logic. Taking into account the previous results we think that the pair HOL/NuPRL is a good choice for the source and the target systems from the viewpoint of these three requirements.

Challenges. There are two major obstacles that we faced in the context of this work: On the theoretical level, the main challenge was to supplement Howe's semantic justification by a proof of soundness on which the translator could be potentially based. The primary concern here is that HOL inference rules, taken literally, are not sound in NuPRL. On the implementation level, a technical difficulty has been the lack of proof objects in HOL. Hence, it was necessary to equip the HOL system with an explicit notion of proof objects and support for exporting such objects.

Outline of the Paper. We begin in Section 2 by giving a brief introduction to the logical foundations of HOL and NuPRL and their implementations. Section 3 discusses the classical extension of NuPRL that we are using and sketches the translation of formulas and proofs. Section 4 gives a high-level description of the translator and complements it with a discussion of some implementation details

[5] We always use "NuPRL" to refer to Howe's variant of NuPRL which enjoys this hybrid set-theretic/computational semantics. Howe's classical extension of NuPRL is then obtained by adding type-theoretic version of the law of the excluded middle.

and some practical experience that we have found to be interesting. We compare our work with recent work by Ewen Denney [6] on translating proofs from HOL to Coq in Section 5. Finally, we present some conclusions in Section 6.

2 Overview of the Theorem Provers

This section gives a brief introduction to the HOL and NuPRL proof development systems and their underlying formal logics and type theories.

2.1 The HOL System

Logic. HOL [9] is an interactive theorem prover based on a Gentzen-style formalization of higher-order logic over a simply typed lambda calculus extended by Hindley-Milner polymorphism, which admits types with (implicitly quantified) type variables.

Syntactically, HOL clearly distinguishes two kinds of entities: types and terms. Types are built from type variables and type constants, which include type operators and regular type constants:

$$Type := TypeVar \mid TypeConst(Type_1, \ldots, Type_n)$$

HOL terms are constructed from free and bound variables as well as constants using application and typed λ-abstraction:

$$Term := FVar_{Type} \mid BVar \mid Const_{Type} \mid Term\ Term \mid \lambda\ Type\ .\ Term$$

Notice that HOL terms carry explicit type information embedded into them. This is done by specifying free variables and constants together with their types. Since bound variables are implemented using de Bruijn indices [5], bound variables are just natural numbers, but their types can be extracted from the type of the λ-term they refer to. In HOL all terms used in theories and rules are well-typed, and, as a result, the unique type of any HOL term can be recursively computed. In spite of the use of de Bruijn indices in the HOL system we prefer to use names in the following for better readability.

Formulas of HOL, which we also refer to as propositions, are terms of the primitive propositional type `bool`. In addition to this type constant, the inference rules of HOL, which are given in Fig. 1, presuppose only two other primitive constants, namely polymorphic equality and boolean implication. The fact that `bool` is actually the boolean data type with negation, conjunction, disjunction, polymorphic universal and existential quantifiers and a polymorphic version of Hilbert's ϵ-operator is ensured axiomatically by the logical theory of which all user-defined theories are extensions. It is interesting to note that the inference rules which define the formal system of HOL are independent of the logical theory,[6] which is a theory like any other theory. In fact, it is the logical theory which makes the logic of HOL classical.

[6] Although in the HOL implementation we discovered a subtle (and somewhat ugly) dependency of the modus ponens rule on the definition of negation.

$$\frac{}{A \vdash A}\ \text{(ASSUME)} \qquad \frac{}{\vdash (\lambda y.M)N = M[N/y]}\ (\beta\text{-CONV})$$

$$\frac{G \vdash A \quad H \vdash A \Rightarrow B}{H, G \vdash B}\ \text{(MP)} \qquad \frac{H_i \vdash M_i = N_i \quad i \in \{1, \ldots, n\} \quad G \vdash A[\overline{M}/\overline{z}]}{H_1, \ldots, H_n, G \vdash A[\overline{N}/\overline{z}]}\ \text{(SUBST)}$$

$$\frac{H, A \vdash B}{H \vdash A \Rightarrow B}\ \text{(DISCH)} \qquad \frac{H \vdash M = N}{H \vdash (\lambda y : \tau.M) = (\lambda y : \tau.N)}\ \text{(ABS)}$$

$$\frac{}{\vdash M = M}\ \text{(REFL)} \qquad \frac{H \vdash A}{H \vdash A[\overline{\tau}/\overline{\alpha}]}\ \overline{\alpha} \text{ do not occur in } H\ \text{(INST\_TYPE)}$$

Fig. 1. Primitive Inference Rules of HOL

System Design. HOL is implemented in SML [15] and the type `thm` of theorems is defined as an abstract SML type. The only information a theorem contains is essentially its sequent, i.e. the hypotheses and the conclusion. Typechecking ensures that SML functions can only produce elements of `thm` using constructors corresponding to the inference rules of HOL. Such functions have to be executed in order to actually generate a provable theorem, but the evidence of provability is not stored anywhere in the system.

Wai Wong [23] added a proof recording mechanism to HOL.[7] Each time an inference rule is called, all parameters of this rule are passed to a special proof recording function that can be defined at a user's discretion. In our work this proof recording mechanism is used to construct a proof object.

2.2 The NuPRL System

Logic. NuPRL is based on a variant of Martin-Löf's polymorphic and extensional type theory [13, 20], which in contrast to HOL is constructive. NuPRL is a type theory which is naturally equipped with a higher-order logic via a propositions-as-types interpretation. Each formula, which we also refer to as a proposition, can be interpreted as a type, and provability of this formula corresponds to the fact this type is inhabited. Unlike some other type theories such as the Calculus of Constructions [4] the elements of such propositional types should not be interpreted as proofs, since type checking in NuPRL is not decidable, but merely represents the computational contents of the associated proposition.

Both types and their elements are represented in NuPRL by terms. These terms are untyped in the sense that there is in general no effective procedure to reconstruct types of NuPRL terms, which is why NuPRL is sometimes called an "untyped type theory". Syntactically, a NuPRL term is build from so-called abstractions, where each abstraction has an identifier, a list of parameters, and a list of subterms with binders. A subterm with binder is a term with an attached list of variables bound by it. The abstraction identifier (such as `var`, `int`, `add`, `lambda`, etc.) is the main descriptor, but sometimes an abstraction requires extra parameters to be specified. For example, each variable is represented as

[7] Another approach to proof terms for Isabelle/HOL is described in [2].

an abstraction `var` together with its name as a parameter. As one can see, the type information is not a fixed part of the NuPRL term structure, and reflecting the polymorphic nature of NuPRL, some terms, such as the λ-abstraction $\lambda x.x$, represent elements of many different types. The NuPRL abstraction mechanism is a powerful tool, which in connection with NuPRL's user-definable display forms, provides syntactic flexibility and readability. In most cases abstractions are introduced by specifying them through existing ones, but it is also possible to introduce an abstraction that is not associated with a definition, in which case we refer to it as a primitive abstraction. A special case of abstractions are constants, which do not have any associated subterms.

The flexibility of NuPRL's polymorphism and the fact that NuPRL does not impose computability of types in any sense enables NuPRL to use a rich and open-ended type theory, which has a predicative cumulative hierarchy of type universes, dependent function types, subset types, parametrized inductive types, quotient types, and other features not present in HOL. As one would expect, most features of NuPRL are not essential for representing HOL theories. In fact, only a few abstractions and inference rules of NuPRL will be used by our proof translator. A detailed description of a fragment of NuPRL sufficient for our work can be found in [21].

System Design. NuPRL is implemented in a hybrid programming framework of Common Lisp and Classic ML. Core parts of the system are written in Lisp, and ML is mostly used as the language to express theorem proving tactics. Data structures for NuPRL terms and proofs are defined in Lisp but are accessible via an ML interface. As a result, dealing with the Lisp part of NuPRL is rarely necessary. In fact, we have implemented the main part of the proof translator entirely in Classic ML on the top of NuPRL.

Unlike HOL, NuPRL uses an explicit representation of proofs as trees. A proof of a theorem consists of the theorem itself, a justification which allows us to infer this theorem, and a list of subproofs, containing one proof for each premise needed by the justification. The justification can either be an instance of a NuPRL inference rule, or, more generally, a NuPRL tactic. To emphasize the style of backward-reasoning employed in NuPRL, we refer to the theorems of the proof and its subproofs also as the goal and its subgoals, respectively.

3 Logical Foundations of the Translator

In [21] we have shown that the mapping from HOL to the classical extension of NuPRL can be understood in the framework of general logics [14] as a composition of two stages: The first stage is a *translation* of an *axiomatic theory of HOL* into an *axiomatic theory of the classical extension of NuPRL*. The use of the term "axiomatic" emphasizes the fact that the theories are not necessarily only definitional extensions of the base logic. The second stage is the *interpreta-*

tion of an axiomatic theory inside the classical extension of NuPRL.[8] Whereas the translation stage is of meta-logical nature, the interpretation stage can take place inside the logic of NuPRL in a formally rigorous way. However, since there are many possible choices for the interpretation of translated HOL theories in NuPRL theories, and certain proofs, namely of the NuPRL axioms resulting from the translation of HOL axioms, to be carried out, the user might need to intervene before each separate HOL theory is translated.

A Classical Extension of NuPRL. Unlike HOL, NuPRL is based on a constructive type theory. Hence, valid HOL theorems like the law of the excluded middle $\forall P.P \vee \neg P$ and logical extensionality $\forall P.(P \Leftrightarrow Q) \Leftrightarrow (P = Q)$ are not provable in NuPRL. There are at least three potential ways to set up the translation:

- First, one can look for a translation that maps HOL into the constructive type theory of NuPRL. For instance, Gödel's double-negation translation (see, for example, [16], p.98) is known to do just that. Unfortunately, double negation or any similar translation would not be very useful for connecting theorem proving systems. The results translated under a non-trivial mapping are difficult to use as lemmas in the target system. Notice also that this translation would still require logical extensionality as an axiom.
- Second, one can consider a stronger non-conservative extension of the constructive type theory of NuPRL by the axiom of the excluded middle, and consider the most straightforward mapping of HOL in this classical extension of NuPRL. Under such a mapping, HOL propositions are translated into NuPRL propositions and logical HOL operators into their NuPRL counterparts. Unfortunately, this approach does not lead to a general solution. The difficulty is that HOL is an impredicative higher-order logic, but NuPRL has a predicative higher-order logic, which is inherited from its predicative type theory via the Curry-Howard isomorphism. As a consequence, the propositional type `bool` of HOL is closed under universal quantification, whereas in NuPRL the use of universal quantification over a proposition in $\mathbb{P}_i$ can result in a proposition from a higher propositional universe $\mathbb{P}_{i+1}$. Hence, there is no single propositional universe in NuPRL that is a suitable image of HOL's propositional type `bool` under this translation. Still, this partial solution could be useful to translate certain predicative developments from HOL into NuPRL.[9] On the other hand, such an approach provides a simple and general solution for the embedding of HOL into an impredicative type theory such as the Calculus of Inductive Constructions (cf. Section 5).
- Third, following Howe [12] one can translate HOL's propositional type `bool` into the boolean type $\mathbb{B}$ of NuPRL. The type $\mathbb{B}$ is not one of the NuPRL

[8] By means of an interpretation we can often obtain a *computationally meaningful* theory which makes only use of the constructive sublanguage of the classical extension of NuPRL that we consider in this paper.

[9] Notice, however that we cannot assume logical extensionality in NuPRL, since this would be inconsistent with NuPRL's nontrivial data types, which cannot be distinguished from propositional types.

primitive types, but is defined as a disjoint union of two single-element types, and all the operators of boolean logic and their (classical) properties are derived, including the law of the excluded middle and logical extensionality. Clearly, $\mathbb{B}$ is a model of classical boolean logic inside NuPRL's constructive type theory, and therefore an obvious candidate for the image of the HOL type `bool`. The only problem is that it lacks some of the features needed for the translation, namely a polymorphic boolean-valued equality, a polymorphic boolean-valued universal quantifier, and a polymorphic version of Hilbert's ε-operator which uses boolean-valued predicates.

In our work we have followed the third approach, since it seems to be the best solution in terms of usability and generality. To fill the remaining gap between the propositional type `bool` of HOL and the boolean type $\mathbb{B}$ of NuPRL we extended NuPRL by new primitive abstractions and axioms as follows.

– HOL equality is a boolean predicate, unlike the standard NuPRL equality $x =_T y$ which is a type living in some propositional univese $\mathbb{P}_i$. In order to translate HOL equality statements into NuPRL terms of type $\mathbb{B}$, we define in NuPRL a new primitive abstraction $x =^b_T y$ which stands for boolean-valued equality of elements of a type T. The following two axioms, that we have added to NuPRL, state that we have a boolean-valued equality that precisely mirrors NuPRL's propositional equality:

$$\forall T : \mathbb{U}_i.\ \forall x, y : T.\ x =^b_T y \in \mathbb{B}$$

$$\forall T : \mathbb{U}_i.\ \forall x, y : T.\ (\uparrow (x =^b_T y)) \Leftrightarrow x =_T y$$

(where $\uparrow b$ stands for the proposition $b =_{\mathbb{B}} true$)
– We dealt similarly with the universal quantifier which has type $\mathbb{P}_i$ in NuPRL, i.e. we introduced a boolean counterpart $\forall_b$ with the following axioms:

$$\forall T : \mathbb{U}_i.\ \forall b : T \to \mathbb{B}.\ (\forall_b x : T.\ b(x)) \in \mathbb{B}$$

$$\forall T : \mathbb{U}_i.\ \forall b : T \to \mathbb{B}.\ (\uparrow \forall_b x : T.\ b(x)) \Longleftrightarrow (\forall x : T \uparrow b(x))$$

– Finally, we added a non-constructive ε-operator in order to interpret the corresponding HOL operator. The axiom for this operator is:

$$\forall T : \mathbb{U}.\ \forall b : T \to \mathbb{B}.\ T \Rightarrow (\varepsilon x : T.\ b(x)) \in T$$

For the understanding of the last axiom recall that NuPRL is based on propositions-as-types paradigm and, as a result, any type T is simultaneously a proposition which asserts non-emptiness of this type.

In [12] it is shown that the operators and axioms given above can all be derived from a simple extension of NuPRL by the axiom of the excluded middle $\forall P.P \vee \neg P$ (more precisely, its type-theoretic version). Assumption of this axiom in NuPRL means, according to the propositions-as-types interpretation, that the corresponding dependent type is inhabited by a function which "decides" for each type P if P is inhabited and returns one of its elements if this is the case. In fact, Howe uses this axiom to define a function $\downarrow_b$ that casts propositions into booleans, and then defines boolean equality and the boolean universal quantifier by lifting their NuPRL counterparts using $\downarrow_b$. Another function, that can be obviously derived from the axiom of the law of the excluded middle, selects an element from a given non-empty type. Howe uses such a function to define the ε-operator.

Translation of Formulas. The translation of formulas of HOL into propositions of NuPRL is a straightforward recursive procedure once HOL constants are mapped into corresponding NuPRL constants. The mapping of constants, which we call a *dictionary*, has to be provided by the user. The translation is then a natural extension of this mapping to a mapping from the types and terms of HOL into the terms of NuPRL: Type operators of HOL are mapped to functions operating on type universes. Typed λ-abstractions of HOL are translated to λ-abstractions in NuPRL (although the latter are untyped, we decided to keep the type information for better readability). Furthermore, implicitly polymorphic functions (and their types) in HOL are translated to explicitly polymorphic functions (and corresponding types) in NuPRL, and suitable explicit type arguments are generated on the NuPRL side when such functions are applied.

The proof translator requires that the dictionary is an injective mapping of HOL constants into NuPRL constants.[10] For example, the HOL type `bool` is mapped to the NuPRL type $\mathbb{B}$ by an entry

$$\alpha(bool) = \mathbb{B} \ .$$

In some cases we would like to map an HOL constant to a more complicated NuPRL term. For instance, the HOL conjunction $\wedge$ is a function of type `bool` $\rightarrow$ `bool` $\rightarrow$ `bool`, whereas in NuPRL conjunction $\wedge$ is an abstraction with two subterms but not a constant. Hence, we would like to have a translation:

$$\alpha(\wedge) = \lambda p, q.p \wedge q$$

To this end we exploit the flexibility provided by the interpretation stage mentioned before, that is we introduce an explicit NuPRL counterpart for each HOL constant, which in this case would be a new NuPRL constant $\wedge_h$ and we interpret this constant by adding a definition $\wedge_h := \lambda p, q.p \wedge q$. That makes the corresponding dictionary entry a simple renaming, namely $\alpha(\wedge) = \wedge_h$, and enhances readability of the result of the translation. In a similar way we deal with other HOL constants such as $\vee_h$, $\rightarrow_h$, $=_h$, etc.

After an HOL formula is translated into a NuPRL formula it becomes a term of type $\mathbb{B}$. In order to treat this term as a NuPRL theorem, we have to cast it into a propositional type. To this end, we use the standard NuPRL function

[10] In the theoretical treatment [21] we have abstracted from renaming issues by using names of HOL constants directly for NuPRL constants.

assert $\uparrow$, which is defined as $\uparrow b := (b =_{\mathbb{B}} true)$. In addition, since, unlike HOL, NuPRL does not allow free variables in theorem statements, we bind all such variables by an appropriate universal quantifier. For instance, the HOL theorem $x_\sigma = x_\sigma$ (provable in one step using the REFL rule), is translated into the NuPRL statement

$$\forall \alpha(\sigma) : \mathsf{S}.\, \forall x : \alpha(\sigma) \quad \uparrow (=_h \ x\ x)$$

where S is a type of all representations of HOL types.[11] Note that in most cases translated HOL theorems can be lifted to NuPRL propositions by propagating $\uparrow$ through boolean quantifiers, connectives, and equalities. This transforms these boolean operations into their regular NuPRL (propositional) counterparts. We have developed a NuPRL tactic to perform this transformation.

Translation of Inference Rules. HOL inference rules, taken literally, are not valid in the classical extension of NuPRL, because they are missing explicit well-formedness conditions. These conditions are implicit in HOL because a built-in type checking mechanism guarantees that each HOL term is well-formed. Because of its open-ended type system, NuPRL imposes much weaker a priori conditions on its terms. In fact, any syntactically valid expression is a term in NuPRL. In the inference rules of NuPRL well-formedness is usually enforced through extra premises which give rise to so-called well-formedness subgoals during a proof. For the standard types of NuPRL these subgoals can be proven automatically by NuPRL's `Auto` tactic in the majority of cases.

To translate inference rules from HOL into NuPRL we have to come up with *derived* rules in NuPRL that match HOL rules as closely as possible and have suitable well-formedness subgoals to reflect the fact that only well-typed terms occur in the rule. Our proof translator redirects well-formedness subgoals to the `Auto` tactic. Although in general these extra subgoals might not be provable by Auto tactic (or even false), in the case when the derived rules are applied to the translation of HOL formulas, well-formedness subgoals are provable by a slightly modified version of the `Auto` tactic.

4 Implementation of the Proof Translator

Adding Proof Objects to HOL. The generation of new HOL theorems from existing ones is done in ML by HOL inference rules which are ML functions with an arbitrary number of arguments of type `thm` and a result of the same type. Many actual proofs are written in terms of tactics which are, essentially, ML programs combining several applications of inference rules.

Once an ML term of type `thm` is parsed by the system, verified by the type checker, and executed successfully, the value of the term (representing the the-

[11] In order to interpret the HOL axiom for the ε-operator in NuPRL, all translations of HOL types have to be nonempty. In the present version of the proof translator we ignore nonemptiness condition and just use $\mathbb{U}_i$ for S (instead of the subtype of non-empty types described in [12, 21]), since most HOL proofs do not depend on this condition. We think that in the future the proof translator should support both possibilities.

orem) becomes available and can be used as a lemma in other proofs, but there is no way to reconstruct the proof from the resulting theorem.

Clearly, in order to translate HOL proofs into NuPRL we first have to recover those proofs. Our initial idea was to extract the proofs from the files containing the ML code that generated them. Unfortunately, this is not an easy task. Since HOL tactics could potentially make use of the full power of Standard ML, which is the tactic-language of HOL, the proof translator would need to include a general compiler or interpreter from Standard ML into Classic ML, which is used in NuPRL as the tactic language.

A simpler approach, that we have eventually adopted, is to modify the HOL abstract data type `thm` in such a way that every element of this type stores additional information about how it has been constructed. To achieve this, we have changed the HOL definition of the `thm` type from

```
datatype thm = THM of Term.term list * Term.term
```

to

```
datatype thm = THM of Term.term list * Term.term
                        * (string * just_arg list)
```

where an element of type string represents the name of the inference rule that created this theorem and the list over `just_arg` stores additional arguments that determine the instance of the inference rule which was actually used. For example, if a theorem $H \vdash (\lambda y.M) = (\lambda y.N)$ was proven using the HOL abstraction rule

$$\frac{H \vdash M = N}{H \vdash (\lambda y.M) = (\lambda y.N)}\ (\text{ABS})$$

then it will have ABS as the name of the inference rule and the variable y and theorem $H \vdash M = N$ as its arguments[12]. We have modified all HOL theorem constructors that operate on the type `thm` to work correspondingly. Of course, once each theorem keep track of its parents theorems, the entire proof tree can be reconstructed recursively.

After these modifications to HOL, we have recompiled the system to make every theorem in every HOL theory aware of its proof. Although, this modification drastically increases the size of `thm` objects, we have not observed a noticeable degradation in the system performance.

Translating Rules as Tactics. Since HOL and NuPRL have different sets of primitive inference rules, typically several NuPRL rules have to be used to imitate one HOL rule. In the previous section we have expressed this idea by saying that there are derived NuPRL rules that closely match HOL rules. Technically, since NuPRL does not have an explicit concept of derived rules, we have written special NuPRL tactics that combine several primitive inference rule to mimic one derived rule application.

[12] We are using Wong's [23] `just_arg` type to store inference rule arguments.

In most cases it was possible to define such tactics using existing NuPRL tactics such as the backchaining tactic `BLemma` together with a suitable theorem to model HOL inference rules. For example, the HOL inference rule

$$\frac{H, A \vdash B}{H \vdash A \Rightarrow B}\ (\mathrm{DISCH})$$

is imitated by our translator essentially via backchaining through the lemma

$$\forall p, q : \mathbb{B}.\ (\uparrow p \Rightarrow \uparrow q) \Rightarrow \uparrow (\Rightarrow_h\ \ p\ q)$$

followed by a decomposition of implication using the tactic `D 0` and a call to NuPRL's `Auto` tactic to eliminate potential well-formedness subgoals as discussed earlier:

```
let DISCH = BLemma 'DISCH_lemma' THENM D 0 THENW Auto;;
```

Dealing with Derived Rules of HOL. Our work was substantially complicated by the status of HOL derived rules. In spite of Gordon's original intention [9] that all the HOL derived rules are reduced to primitive inference rules, in the version 90.10 of HOL that we have used the majority of originally derived rules are represented in the HOL system as new constructors for the `thm` type.[13] Hence, the majority of derived rules are essentially proclaimed to be new primitive rules. Our understanding is that this change has been done at some point in the past for efficiency reasons.

To face the reality of the HOL implementation we decided to construct NuPRL equivalents for these derived rules the same way as we have done it for primitive rules. On the positive side, the higher level of abstraction provided by the derived rules allows us to avoid reducing logical operators (in particular conjunction and the existential quantifier) to terms involving the ϵ-operator. This does not only lead to more readable proofs, but more importantly the ϵ-operator is only needed in cases where it is explicitly used in a theory.[14]

Implementation Details. The proof translator is implemented as a pair of functions (`export`, `import`) in two different dialects of ML. The function `export` is written in Standard ML and is integrated into the HOL system. When this function is called in the HOL environment on any theorem named `theorem`, a separate file `theorem.ml` is created and a description of the theorem's proof is stored in the file in what we call *portable theorem format*. Essentially, this file contains a theorem together with its associated proof tree written in Classic ML. When the `theorem.ml` file is loaded into the NuPRL environment this tree is

[13] Although the HOL system contains deactivated ML code that is supposed to define derived rules via primitive rules, an inspection of this code indicated that it is no longer compatible with the current version of HOL.

[14] The current version of our proof translator can deal with nearly all basic inference rules and most of the derived rules mentioned above. In this point we are extending the theoretical treatment [21] which uses the HOL logic as presented in [9].

re-created as a Classic ML data structure. The second function `import`, written in Classic ML, takes this structure as an argument, translates the theorem into a NuPRL proposition, proves it by invoking the NuPRL tactics corresponding to the HOL inference rules given by the proof, and adds the new result to the NuPRL library.

Although the translation process itself does not require user intervention, it relies on the user-defined dictionary in order to translate HOL constants into their NuPRL counterparts. The dictionary not only allows the user to choose meaningful names for the translation of HOL constants, but it is also needed to avoid name clashes which can occur, since in the current version (4.2) NuPRL has a flat name space, and the names for the translated constants could have been already used in one of the theories the user is working with.

Theorem Name	**Statement**	**HOL rule invocations**
DISJ_ASSOC	$\forall A, B, C.\ A \vee B \vee C = (A \vee B) \vee C$	3218
DE_MORGAN_THM	$\forall A, B.\ (\neg(A \wedge B) = \neg A \vee \neg B) \wedge$	
	$\wedge(\neg(A \vee B) = \neg A \wedge \neg B)$	4733
LEFT_AND_OVER_OR	$\forall A, B, C.\ A \wedge (B \vee C) = A \wedge B \vee A \wedge C$	4215
EQ_EXPAND	$\forall t1, t2.\ (t1 = t2) = t1 \wedge t2 \vee \neg t1 \wedge \neg t2$	2618
COND_ABS	$\forall b, f, g.\ (\lambda x.b \Rightarrow (fx)\vert(gx)) = (b \Rightarrow f\vert g)$	1368
COND_RATOR	$\forall b, f, g, x.\ (b \Rightarrow f\vert g)x = (b \Rightarrow (fx)\vert(gx))$	1357

Fig. 2. Sample HOL proof sizes

Efficiency Issues. Although the translated proofs that the proof translator produced in a number of small examples of logical nature are valid NuPRL proofs, we were surprised by their sizes. In many cases, a few lines of HOL proofs, invoking just a few tactic calls, turned out to hide thousands of HOL inference rule applications (see Figure 2). Since every HOL inference rule call is translated into a NuPRL tactic call, which itself normally requires several inference rule applications, the resulting NuPRL proofs even for basic propositional theorems are unexpectedly long. From our experience, it takes NuPRL several minutes just to check the validity of the produced proof.

5 Related Work

The translation of proofs between higher-order logic theorem provers is a relatively new field which has both practical potential and interesting theoretical aspects. Apart from our own work [21, 22], which heavily draws on earlier work by Howe [12], we are only aware of one further implementation of a higher-order logic proof translator, namely the translator from HOL into Coq [1] developed by Ewen Denney and described in [6]. Another piece of work which is related to ours from an implementation point of view is the work by Wai Wong [23] on rechecking HOL proofs using an external HOL proof checker. Our implementation makes use of the basic infrastructure that he provides, but we use our own internal and external proof format to facilitate the connection with NuPRL.

Concerning the proof translator from HOL to Coq [6] there are a number of similarities and differences to our work that we briefly discuss here. Both Coq and NuPRL are goal-oriented theorem provers based on expressive type theories and following the propositions-as-types paradigm. Coq is based on the Calculus of Inductive Constructions with a number of extensions. In contrast to the rich open-ended, predicative and polymorphic type theory of NuPRL [3], the Calculus of Constructions [4] is a monomorphic and impredicative type theory with dependent function types as the central concept, and the Calculus of Inductive Constructions [1] extends it by a predicative universe hierarchy and inductive definitions.

Coq uses the impredicative universe `Prop` for logical propositions, and due to its impredicativity the HOL/Coq translator can directly use this universe to represent the propositional type `bool` of HOL. Consequently, [6] assumes the axiom of the excluded middle for propositions in `Prop`. This is quite different from our approach, which clearly separates HOL propositions from NuPRL propositions, so that we are actually using NuPRL as a hybrid logic. Since, the Coq/HOL translator is already based on a non-conservative extension of Coq, one would expect that it also assumes logical extensionality of propositions in `Prop` to mirror logical extensionality of `bool`. However, [6] explicitly avoids this axiom by transforming boolean equality of HOL into an equivalence on Coq propositions, which is somewhat unsatisfactory, since the HOL type `bool` does not only represent propositions, but more importantly the well-known boolean data type, and `Prop` does not have the status of a data type in Coq due to the lack of suitable elimination principles over uninformative types. Although [6] does not give any meta-logical results to justify the design choices, we believe that in particular in view of the treatment of HOL equality, the translation deserves a careful study, since functions in HOL clearly respect equality, but the corresponding equivalence in Coq does not in general enjoy the Leibnitz property.[15]

In spite of these foundational differences, we found many similarities on the practical side, namely that both translators make use of an intermediate portable proof format for exchanging proofs between the two provers, and they both use mostly backchaining via suitable lemmas to model applications of HOL primitive inference rules. A unique feature of the HOL/Coq translator is the use of the DAG representation of the proof in the intermediate format and an automated "factoring out" of lemmas in the final proof. We think that this is an interesting idea worth considering also for the HOL/NuPRL translator which could partially solve the inefficiency issues we have discussed before.

Apart from the differences in the choice of the theorem provers and the mapping, there is a more important methodological difference between these two lines of research. Our starting point was the semantic justification that Howe gave for the mapping from HOL into NuPRL [12] together with our proof-theoretic

[15] An issue that remains unclear in [6] is how non-emptiness of HOL types is reflected. In practice, we found that most HOL proofs do not need this assumption, although it is needed for the interpretation of the logical theory of HOL.

soundness proof for this mapping [21] from which we essentially derived the proof translator. The work [6], however, neither gives a corresponding soundness proof for the mapping from HOL to Coq nor a semantic justification. We think that in general such meta-logical justifications are important to obtain a better understanding of the translation mapping. On the other hand, since Coq is based on a monomorphic and impredicative type theory in contrast to NuPRL which is polymorphic and predicative, the intuitive distance between HOL and Coq is smaller than the distance between HOL and NuPRL, both from a type-theoretic and from a semantic point of view, which makes such meta-theoretical justifications probably more critical for the HOL/NuPRL translator.

6 Conclusions

We have discussed the design principles and the implementation of a proof translator from HOL to NuPRL. The design of the translator is based on our mathematical proof of soundness for a mapping of HOL theories to NuPRL theories [21]. The mapping at the level of theories agrees with Howe's translation [12], but our justification is proof-theoretic whereas Howe's is semantic. Both approaches complement each other. Furthermore, the proof-theoretic nature of our result provides a method for the translation of HOL proofs into NuPRL proofs. In summary, we have designed and implemented a translator based on strong meta-logical justifications, both semantic (thanks to Howe) and proof-theoretic (our own) which in addition generates proofs as internal justifications in the target logic. Further work is required to cover some additional inference rules and to reduce the size of proofs, so that translation of large developments becomes feasible.

Acknowledgements. We would like to thank Robert Constable and Stuart Allen for several discussions on the particularities of NuPRL and the issue of logic translation. We also would like to thank Doug Howe, since this project draws on his earlier work and would not have been possible without his support. He made not only the source code of the HOL/NuPRL connection available to us, but he also gave us the opportunity to discuss with him our initial ideas on a proof-theoretic soundness result.

References

1. B. Barras et al. The Coq Proof Assistant Reference Manual: Version 6.1. Technical Report RT-0203, INRIA, May 1997.
2. S. Berghofer and T. Nipkow. Proof terms for simply typed higher order logic. In M. Aagaard and J. Harrison, editors, *13th International Conference on Theorem Proving in Higher Order Logics*, volume 1869 of *Lecture Notes in Computer Science*, pages 38–52, Portland, Oregon, August 2000. Springer.
3. R.L. Constable et al. *Implementing Mathematics with Nuprl Proof Development System*. Prentice Hall, 1986.
4. T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, 1988.

5. N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indag. Math.*, 34:381–392, 1972.
6. E. Denney. A Prototype Proof Translator from HOL to Coq. In M. Aagaard and J. Harrison, editors, *The 13th International Conference on Theorem Proving in Higher Order Logics*, volume 1869 of *Lecture Notes in Computer Science*, pages 108–125, Portland, Oregon, August 2000. Springer-Verlag.
7. A.P. Felty and D.J. Howe. Hybrid interactive theorem proving using Nuprl and HOL. In W. McCune, editor, *Automated Deduction – CADE-14*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 351–365, Berlin, 1997. Springer-Verlag.
8. J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
9. M.J.C. Gordon and T.F. Melham. *Introduction to HOL – A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.
10. D. J. Howe. A classical set-theoretic model of polymorphic extensional type theory. Manuscript.
11. D.J. Howe. Importing mathematics from HOL into Nuprl. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics*, volume 1125 of *Lecture Notes in Computer Science*, pages 267–282, Berlin, 1996. Springer-Verlag.
12. D.J Howe. Semantics foundation for embedding HOL in Nuprl. In M. Wirsing and A. Nivat, editors, *Algebraic Methodology and Software Technology*, volume 1101 of *Lecture Notes in Computer Science*, pages 85–101, Berlin, 1996. Springer-Verlag.
13. P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, Napoli, 1984.
14. J. Meseguer. General logics. In H.-D. Ebbinghaus et al., editor, *Logical Colloquium, 1987*, pages 275–329. North-Holland, 1989.
15. R. Milner, M. Tofte, R.M. Harper, and D.B. MacQueen. *The Definition of Standard ML (Revised).* MIT Press, Cambridge, 1997.
16. G. Mints. *A Short Introduction to Intuitionistic Logic.* Kluwer Academic/Plenum Publishers, 2000.
17. P. Naumov. Importing Isabelle Formal Mathematics into NuPRL. In *Supplemental proceedings of the 12th International Conference on Theorem Proving in Higher Order Logics*, Nice, France, September 1999.
18. P. Naumov. Formalization of Isabelle Meta Logic in NuPRL. In *Supplemental proceedings of the 13th International Conference on Theorem Proving in Higher Order Logics*, pages 141–156, Portland, OR, August 2000.
19. L.C. Paulson. *Isabelle – A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1994.
20. K. Petersson, J. Smith, and B. Nordstroem. *Programming in Martin-Löf's Type Theory. An Introduction.* International Series of Monographs on Computer Science. Oxford: Clarendon Press, 1990.
21. M.-O. Stehr, P. Naumov, and J. Meseguer. A proof-theoretic approach to HOL-Nuprl connection with applications to proof translation (full version). http://cs.hbg.psu.edu/˜naumov/Papers/holnuprl.ps, March 2000.
22. M.-O. Stehr, P. Naumov, and J. Meseguer. A proof-theoretic approach to HOL-Nuprl connection with applications to proof translation. In *15th International Workshop on Algebraic Development Techniques*, Genova, Italy, April 2001. To appear. See [21] for the full version.
23. W. Wong. Validation of HOL proofs by proof checking. *Formal Methods in System Design: An International Journal*, 14(2):193–212, 1999.