An Environment For Automated Reasoning About Partial Functions *

David A. Basin

Department of Computer Science,
Cornell University, Ithaca, NY 14853

Abstract

We report on a new environment developed and implemented inside the Nuprl type theory that facilitates proving theorems about partial functions. It is the first such automated type-theoretic account of partiality. We demonstrate that such an environment can be used effectively for proving theorems about computability and for developing partial programs with correctness proofs. This extends the well-known proofs as programs paradigm to partial functions.

Key words and phrases. Automated program development, computability, constructivity, partial functions, tactics, theorem proving, type theory, unsolvability.

1 Introduction

Over the past 20 years, research by Martin-Löf [8], Constable [3], Coquand and Huet [6], and others has demonstrated that constructive type theory provides a useful foundation for theorem proving and program development. The Nuprl proof development system, developed at Cornell [4], has been used to demonstrate that type theory, in practice, provides a rich framework for theorem proving. However, current type theories are inadequate for reasoning about partial functions. Partial functions cannot be typed in full generality and must be approximated as total functions on subsets of their domain of convergence. This approach is problematic as the exact domain often cannot be represented. Also, these theories provide no means for abstract reasoning about termination, nor do they provide fixpoint induction rules. As a result, they are ill suited for proving theorems about partial computations and for developing partial programs.

*This research was supported in part by NSF grant DCR83-03327
Recent work by Constable and Smith has established a theoretical foundation for reasoning about partial objects in type theory. In [5], they present a method of extending the logic of Nuprl to reason about nontermination. A new type constructor, embodied in the bar operator \( \overline{T} \) on type \( T \), is added. \( \overline{T} \) is inhabited by terms that represent computations of elements in \( T \). An inhabitant of \( \overline{T} \) may diverge, but if it terminates, it converges to an inhabitant of \( T \). For example, the type \( \text{int} \rightarrow \text{int} \) corresponds to the standard notion of a partial function space; it is inhabited by functions that take an integer argument and, if they converge on their input, return an integer.

We demonstrate that this partial type theory, along with proof assisting tactics, can be implemented within Nuprl. Once implemented, the resultant partial type environment is surprisingly powerful. We simultaneously gain the ability to give concise proofs of theorems about computation and we can extend the "proofs as programs" paradigm to partial program development via the type-theoretic equivalent of partial correctness reasoning. Hence, we dramatically increase the power of Nuprl, both as a theorem proving system and as a program development system.

In the next section, we describe the environment. In the third section, we show the development of a recursion theoretic proof in this environment. In section 4, we present a new paradigm for program development and provide an example of its use. Finally, we draw conclusions from our research.

2 The Environment

A type theory may be specified by defining the terms of the theory, and an evaluation relation, along with rules defining types, type membership, and type equality. Constable and Smith [12] extend the Nuprl type theory to a partial type theory as follows: The terms of the partial type theory are the terms of the underlying theory, in our case Nuprl, plus a new term \( \text{fix}(f, x. b) \). The evaluation relationship is augmented to reflect the redex/contractum pair shown below.

\[
\text{fix}(f, x. b)(a), \ b[\text{fix}(f, x. b), a/f, x]
\]

The rules for defining types, type membership, and type equality, are augmented with rules for reasoning about the partial types. A complete list of rules for the partial type theory may be found in [12]. Some representative ones, which we present refinement style\(^1\), are given in figure 1.\(^2\) \text{BarCTotality} provides one way

\(^1\text{In the presentation of a rule, the first line contains the goal to which the rule is applied. The "\(\rightarrow\)" symbol is the printable equivalent of the logical turnstile, and } H \text{ represents a (possibly empty) list of hypotheses. After the goal, indented lines contain the subgoals that result from the application of the refinement rule. The reader is referred to [4] for a complete description of Nuprl and refinement style theorem proving.}\)

\(^2\text{As the partial type theory is still in development, the final rules may differ.}\)