A Practical Approach to Partiality – A Proof Based Approach*

Farhad Mehta

Systransis AG – Transport Information Systems,
Bahnhofplatz, P.O. Box 4714,
CH-6304 Zug, Switzerland

Abstract. Partial functions are frequently used when specifying and reasoning about computer programs. Using partial functions entails reasoning about potentially ill-defined expressions. In this paper we show how to formally reason about partial functions without abandoning the well understood domain of classical two-valued predicate calculus. In order to achieve this, we *extend* standard predicate calculus with the notion of *well-definedness* which is currently used to *filter out* potentially ill-defined statements from proofs. The main contribution of this paper is to show how the standard predicate calculus can be extended with a new set of *derived* proof rules that can be used to *preserve* well-definedness in order to make proofs involving partial functions less tedious to perform.

1 Introduction

Partial functions are frequently used when specifying and reasoning about computer programs. Some basic mathematical operations (such as division) are partial, some basic programming operations (such as array look-ups or pointer dereferencing) are partial, and many functions that arise through recursive definitions are partial or possibly non-terminating . Using partial functions entails reasoning about potentially ill-defined expressions (such as $3/0$) in proofs which (as discussed later in §3 and §4) can be tedious and problematic to work with. Providing proper logical and tool support for reasoning in the presence of partial functions is therefore important in the engineering setting. Although the contributions of this paper are theoretical in nature, they result in practical benefits which will be stated later in this section.

The current approaches for explicitly reasoning in the partial setting [6,7,19] are based on three-valued logic where the *valuation* of a predicate is either true, false, or $\mathsf{undefined}$ (for predicates containing ill-defined expressions). They also propose their own 'special-purpose' proof calculi for performing such proofs. Using such a special-purpose proof calculus has the drawback that it differs from the standard predicate calculus. For instance, it may disallow the use of the

* This research was carried out at the ETH Zurich as part of the EU research project IST 511599 RODIN (Rigorous Open Development Environment for Complex Systems).

S. Liu, T. Maibaum, and K. Araki (Eds.): ICFEM 2008, LNCS 5256, pp. 238–257, 2008.

law of excluded middle (to avoid proving '$3/0 = x \vee 3/0 \neq x$'). Such differences make any special-purpose calculus unintuitive to use for someone well versed in standard predicate calculus since, for instance, it is hard to mentally anticipate the consequences of disallowing the law of excluded middle on the validity of a logical statement. Automation too requires additional effort since the well-developed automated theorem proving support already present for standard predicate calculus [22] cannot be readily reused. Additionally, as stated in [8], there is currently no consensus on which is the 'right' calculus to use.

In this paper we present a general methodology for using standard predicate calculus to reason in the 'partial' setting by *extending* it with new syntax and derived rules. We then *derive* one such special-purpose calculus using this general methodology. We call our approach 'proof based' since (in the spirit of [4] that we build on) we do not make any detours through three-valued semantic arguments (which can be found in [8] and [7]), but confine our reasoning to (syntactic) proofs in standard predicate calculus. The novelty of this approach is that we are able to reduce *all* our reasoning (i.e. within our system, as well as about it) to standard predicate calculus, which is both widely understood and has well-developed automated tool support. This approach additionally gives us a theoretical basis for comparing the different special-purpose proof calculi already present(which is done in §8.1), and the practical benefit of being able to exchange proofs and theorems between different theorem proving systems.

The ideas presented in this paper have additionally resulted in providing better tool support for theorem proving in the partial setting within the RODIN development environment [1] for Event-B [3].

Setting. Our practical setting is that of formal system development in Event-B [3]. The development process consists of *modeling* a desired system and *proving* proof obligations arising from it. The logic used in Event-B is set theory built on first-order predicate calculus. A user can define partial functions in this logic. The results presented here are independent of the Event-B method and its set theory. They are equally applicable in many areas where predicate calculus is used to reason in a setting with potentially ill-defined expressions.

Structure. In §2 we define the syntax (§2.1) and proof rules (§2.2) for standard first-order classical predicate calculus with equality (*FoPCe*). In §2.3 we state the forms of reasoning we use in this paper and in §3 we show how partial functions are defined. In §4 we show how to separate the concerns of well-definedness from those of validity by filtering out ill-defined proof obligations using the well-definedness (WD) operator '$\mathcal{D}$' as in [4,7,8]. In §5 we describe $\mathcal{D}$ and state some of its important properties. The *main contribution* of this paper is in §6 where we show how the notion of well-definedness can be integrated into standard predicate calculus. In §6.1 we extend the definition of $\mathcal{D}$ to sequents. We then formally define the notions of a well-defined sequent (§6.2) and a well-definedness (WD) preserving proof rule (§6.3). In §6.4 we derive a proof calculus ($FoPCe_{\mathcal{D}}$) that preserves well-definedness. In §7 we return to the practical issue of filtering and proving proof obligations. In §8 we compare our approach with related work

and show how our approach can be used as a basis to compare the special-purpose proof calculi presented in [6] and [7]. We conclude in §9 by stating what we have achieved and its impact on the RODIN development environment [1] for Event-B.

2 Predicate Calculus

In this section we define the syntax and proof rules for the standard (first-order, classical) predicate calculus with equality that we will use in the rest of the paper.

2.1 Basic Syntax

Basic formulæ Formulæ in first-order predicate calculus can either be predicates (P) or expressions (E). We define the structure of *basic formulæ* as follows:

$$\begin{array}{lcl} P & := & \bot \mid \neg P \mid P \wedge P \mid \forall x.P \mid E = E \mid R(\vec{E}) \\ E & := & x \mid f(\vec{E}) \end{array}$$

Where $\bot$ is the 'false' predicate, x is a variable, $\vec{E}$ is a finite, ordered sequence of expressions, R is a relational predicate symbol, and f is a function symbol. Equality is denoted by the infix binary relational predicate symbol '='.

Sequents. A sequent is a statement we want to prove, denoted '$\mathsf{H} \vdash G$', where H is a finite set of predicates (the hypotheses), and G is a single predicate (the goal).

We extend this basic syntax in §2.2, §5, and §6 using syntactic definitions such as '$\top \mathrel{\widehat{=}} \neg\bot$'. The '$\mathrel{\widehat{=}}$' symbol represents *syntactic* equivalence. It is not itself part of the syntax, but a *meta-logical* connective.

2.2 Proof Rules for *FoPCe*

Here are the rule schemas that define the basic proof calculus for first-order predicate calculus with equality (*basicFoPCe*):

$$\frac{}{\mathsf{H}, P \vdash P}\ hyp \qquad \frac{\mathsf{H} \vdash Q}{\mathsf{H}, P \vdash Q}\ mon \qquad \frac{\mathsf{H} \vdash P \quad \mathsf{H}, P \vdash Q}{\mathsf{H} \vdash Q}\ cut \qquad \frac{\mathsf{H}, \neg P \vdash \bot}{\mathsf{H} \vdash P}\ contr$$

$$\frac{}{\mathsf{H}, \bot \vdash P}\ \bot hyp \qquad \frac{\mathsf{H}, P \vdash \bot}{\mathsf{H} \vdash \neg P}\ \neg goal \qquad \frac{\mathsf{H} \vdash P}{\mathsf{H}, \neg P \vdash Q}\ \neg hyp$$

$$\frac{\mathsf{H} \vdash P \quad \mathsf{H} \vdash Q}{\mathsf{H} \vdash P \wedge Q}\ \wedge goal \qquad \frac{\mathsf{H}, P, Q \vdash R}{\mathsf{H}, P \wedge Q \vdash R}\ \wedge hyp$$

$$\frac{\mathsf{H} \vdash P}{\mathsf{H} \vdash \forall x \cdot P}\ \forall goal\ (x \ \underline{\mathsf{nfin}}\ \mathsf{H}) \qquad \frac{\mathsf{H}, [x := E]P \vdash Q}{\mathsf{H}, \forall x \cdot P \vdash Q}\ \forall hyp$$

$$\frac{}{\mathsf{H} \vdash E = E}\ = goal \qquad \frac{\mathsf{H} \vdash [x := E]P}{\mathsf{H}, E = F \vdash [x := F]P}\ = hyp$$

Syntactic operators. The rules shown above contain occurrences of so-called *syntactic operators* for substitution and non-freeness. The predicate '$[x := E]P$' denotes the syntactic operator for substitution $[x := E]$, applied to the predicate P. The resulting predicate is P, with all free occurrences of the variable x replaced by the expression E. The side condition '$(x \ \underline{\mathsf{nfin}} \ \mathsf{H})$' asserts that the variable 'x' is not free in any of the predicates contained in 'H'. Both these syntactic operators are defined (using '$\widehat{=}$') on the inductive structure of basic formulæ in such a way that they can always be evaluated away. Their formal definitions can be found in [2]. Syntactic operators can be thought of as 'macros' whose repeated replacement always results in a basic formula. Our basic syntax for formulæ therefore does not need to be extended to take the syntactic operators into account.

Derived Logical Operators. The other standard logical operators $\top$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ and $\exists$ can be expressed in terms of the above basic logical connectives using syntactic definitions such as '$\top \mathrel{\widehat{=}} \neg\bot$'. Their corresponding proof rules can then be derived from the above basic proof rules. Both these steps are standard and their details can be found in §3.3 of [17].The resulting syntax and proof rules correspond to the standard first-order predicate calculus with equality. We will refer to this collection of proof rules as the theory (or proof calculus) *FoPCe*.

2.3 Reasoning

There are two forms of reasoning that we use in this paper. The first is *syntactic rewriting* using syntactic definitions such as '$\top \mathrel{\widehat{=}} \neg\bot$', where all occurrences of '$\top$' in a formula may be replaced with '$\neg\bot$', purely on the syntactic level, to get a syntactically equivalent formula. The second is *logical validity* where we additionally appeal to the notion of proof. All proofs done in this paper use the standard predicate calculus *FoPCe*. When we say that a predicate 'P' is provable, we mean that we have a proof of the sequent '$\vdash P$' using *FoPCe*. Most proofs done in this paper rely on both syntactic and logical reasoning. In §5.2 we discuss precautions we need to take when performing such proofs.

3 Defining Partial Functions

In this section we show how a partial function is defined in our mathematical logic. We first present how this can be done in general, and then follow with an example that will be used in the rest of this paper.

3.1 Conditional Definitions

A partial function symbol 'f' is defined using the following *conditional definition*:

$$\frac{}{C^f_{\vec{x}} \ \vdash \ y = f(\vec{x}) \ \Leftrightarrow \ D^f_{\vec{x},y}} \ f_{def}$$

Conditional definitions of the above form can safely be added to *FoPCe* as an axiom, provided:

1. The variable 'y' is not free in the predicate '$C^f_{\vec{x}}$'.
2. The predicate '$D^f_{\vec{x},y}$' only contains the free variables from '$\vec{x}$' and 'y'.
3. The predicates '$C^f_{\vec{x}}$' and '$D^f_{\vec{x},y}$' only contain previously defined symbols.
4. The theorems:
   Uniqueness: $C^f_{\vec{x}} \vdash \forall y, z \cdot\ D^f_{\vec{x},y} \wedge D^f_{\vec{x},z} \ \Rightarrow\ y = z$
   Existence: $C^f_{\vec{x}} \vdash \exists y \cdot\ D^f_{\vec{x},y}$
   must both be provable using *FoPCe* and the previously introduced definitions.

The predicate '$C^f_{\vec{x}}$' is the *well-definedness condition* for 'f' and specifies its domain. For a total function symbol, '$C^f_{\vec{x}}$' is '$\top$'. Provided '$C^f_{\vec{x}}$' holds, f_{def} can be used to eliminate all occurrences of 'f' in a formula in favor of its definition '$D^f_{\vec{x},y}$'. More details on conditional definitions can be found in [4].

3.2 Recursive Definitions

Note that conditional definitions as described above cannot be directly used to define function symbols recursively since the definition of a function symbol '$D^f_{\vec{x},y}$' may not itself contain the function symbol 'f' that it defines, as stated in the third condition above.

It is still possible to define partial functions recursively in a theory (such as the set theory described in [4] and [3]) which supports the applications of functions that are *expressions* (i.e. not plain function symbols) in the theory. Such recursively defined functions are then defined as *constant* symbols (i.e. total function symbols with no parameters). Function application is done using an additional function symbol for function application (often denoted using the standard function application syntax '$\cdot(\cdot)$') with two parameters which are both expressions: the function to apply, and the expression to apply it to. The definition of this function application symbol is conditional. Its well definedness predicate ensures that the its first parameter is indeed a function, and its second parameter is an expression that belongs to the domain of this function. This methodology is described in detail in [4] which also describes (in §1.5) how functions can be defined recursively.

3.3 A Running Example

For our running example, let us assume that our syntax contains the nullary function symbol '0', and the unary function symbol '*succ*', and our theory contains the rules for Peano arithmetic. We may now introduce a new unary function symbol '*pred*' in terms of '*succ*' using the following conditional definition:

$$\frac{}{E \neq 0 \ \vdash\ y = pred(E) \ \Leftrightarrow\ succ(y) = E}\ pred_{def}$$

Defined in this way, '*pred*' is partial since its definition can only be unfolded when we know that its argument is not equal to 0. The expression '$pred(0)$' is still a syntactically valid expression, but is *under-specified* since we have no way

of unfolding its definition. The expression '$pred(0)$' is therefore said to be *ill-defined*. We do not have any way to prove or refute the predicate '$pred(0) = x$'.

The predicates '$pred(0) = pred(0)$' and '$pred(0) = x \lor pred(0) \neq x$' though, can still be proved to be valid in *FoPCe* on the basis of their logical structure. This puts us in a difficult position since these predicates contain ill-defined expressions. The standard proof calculus *FoPCe* is therefore not suitable if we want to restrict our notion of validity only to sequents that do not contain ill-defined formulæ.

4 Separating WD and Validity

Since our aim is to still be able to use *FoPCe* in our proofs, we are not free to change our notion of validity. We instead take the pragmatic approach of *separating* the concern of validity from that of well-definedness and require that both properties hold if we want to avoid proving potentially ill-defined proof obligations. In this case, we still allow the predicate '$pred(0) = pred(0)$' to be proved to be valid, but we additionally ensure that it cannot be proved to be well-defined. When proving a proof obligation '$\mathsf{H} \vdash G$' we are then obliged to prove two proof obligations:

$\mathsf{WD}: \quad \vdash \mathcal{D}(\mathsf{H} \vdash G)$	$\mathsf{Validity}: \quad \mathsf{H} \vdash G$

The first proof obligation, WD, is the well-definedness proof obligation for the sequent '$\mathsf{H} \vdash G$'. It is expressed using the well-definedness (WD) operator $\mathcal{D}$ that is introduced in §5 and defined for sequents in §6.1. The second proof obligation, Validity, is its validity proof obligation. An important point to note here is that *both these proof obligations may be proved using FoPCe*.

Proving WD can be seen as *filtering out* proof obligations containing ill-defined expressions. For instance, for '$\vdash pred(0) = pred(0)$' we are additionally required to prove '$\vdash 0 \neq 0 \land 0 \neq 0$' as its WD (this proof obligation is computed using definitions that appear in §5 and §6.1). Since this is not provable, we have filtered out (and therefore rejected) '$\vdash pred(0) = pred(0)$' as not being well-defined in the same way as we would have filtered out and rejected '$\vdash 0 = \varnothing$' as not being well-typed. Well-definedness though, is undecidable and therefore needs to be proved. Figure 1 illustrates how well-definedness can be thought of as an additional *proof-based* filter for mathematical texts.

When proving Validity, we may then additionally assume that the initial sequent '$\mathsf{H} \vdash G$' is well-defined. The assumption that a sequent is well-defined can

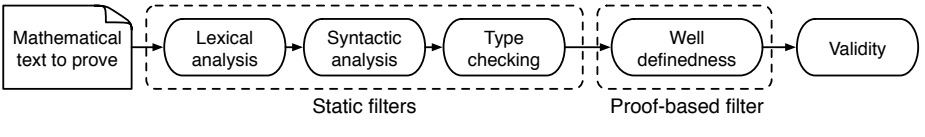

Fig. 1. Well-definedness as an additional filter

be used to greatly ease its proof. It allows us to avoid proving that a formula is well-defined every time we want to use it (by expanding its definition, or applying its derived rules) in our proof. For instance we may apply the simplification rule '$x \neq 0 \vdash pred(x + y) = pred(x) + y$' without proving its premise '$x \neq 0$'. This corresponds to the way a mathematician works.

In §6.4 we show this key result formally; i.e. how *conditional definitions become 'unconditional'* for well-defined sequents.

For the moment though, we may only assume that the initial sequent of Validity is well-defined. In order to take advantage of this property throughout a proof we need to use proof rules that *preserve* well-definedness. Preserving well-definedness in an interactive proof also has the advantage of preventing the user from introducing possibly erroneous ill-defined terms into a proof. A proof calculus preserving well-definedness is presented in §6. Before that, in the next section, we first describe the well-definedness operator.

5 The Well-Definedness Operator

The WD operator '$\mathcal{D}$' formally encodes what we mean by well-definedness. $\mathcal{D}$ is a syntactic operator (similar in status to the substitution operator '$[x := E]$' seen in §2.2) that maps formulæ to their well-definedness (WD) predicates. We interpret the predicate denoted by $\mathcal{D}(F)$ as being valid iff F is well-defined. The $\mathcal{D}$ operator is attributed to Kleene [16] and also appears in [4,7,8], and as the 'δ' operator in [6,11].

Since $\mathcal{D}$ has been previously well studied, we only give in this section an overview of the properties of $\mathcal{D}$ from [4] that we use later in this paper. In §5.1 we define $\mathcal{D}$ for formulæ in *basicFoPCe*. In §5.2 we derive equivalences that allow $\mathcal{D}$ for all formulæ in *FoPCe* to be computed, and state an important properties of $\mathcal{D}$ that we use later in §6.

5.1 Defining $\mathcal{D}$

$\mathcal{D}$ is defined on the structure of formulæ in *basicFoPCe* using syntactic definitions. For expressions, $\mathcal{D}$ is defined as follows:

$$\mathcal{D}(x) \mathrel{\widehat{=}} \top \tag{1}$$

$$\mathcal{D}(f(\vec{E})) \mathrel{\widehat{=}} \vec{\mathcal{D}}(\vec{E}) \wedge C^f_{\vec{E}} \tag{2}$$

where $\vec{\mathcal{D}}$ is $\mathcal{D}$ extended for sequences of formulæ (i.e. $\vec{\mathcal{D}}(\,) \mathrel{\widehat{=}} \top$, $\vec{\mathcal{D}}(F, \vec{F}) \mathrel{\widehat{=}} \mathcal{D}(F) \wedge \vec{\mathcal{D}}(\vec{F})$). An occurrence of a variable in a formula is always well-defined. The occurrence of a function application is well-defined iff all its operands are well-defined (i.e. $\vec{\mathcal{D}}(\vec{E})$ holds), and the well-definedness condition '$C^f_{\vec{E}}$' from the conditional definition of f holds. The resulting definition for the running example '*pred*' is $\mathcal{D}(pred(E)) \mathrel{\widehat{=}} \mathcal{D}(E) \wedge E \neq 0$.

Similarly, $\mathcal{D}$ for $\bot$, $\neg$, $=$ and relational predicate application is defined as follows:

$$\mathcal{D}(\bot) \mathrel{\widehat{=}} \top \tag{3}$$
$$\mathcal{D}(\neg P) \mathrel{\widehat{=}} \mathcal{D}(P) \tag{4}$$
$$\mathcal{D}(E_1 = E_2) \mathrel{\widehat{=}} \mathcal{D}(E_1) \wedge \mathcal{D}(E_2) \tag{5}$$
$$\mathcal{D}(R(\vec{E})) \mathrel{\widehat{=}} \vec{\mathcal{D}}(\vec{E}) \tag{6}$$

Note that we regard relational predicate application as always being total. In case we require partial relational predicate symbols, they can be supported in the same way as partial function symbols.

Since we would like predicates such as '$x \neq 0 \wedge \mathit{pred}(x) = x$' (or similarly, '$x \neq 0 \Rightarrow \mathit{pred}(x) \neq x$') to be well-defined special care is taken while defining the well-definedness of $\wedge$ and $\forall$ as follows:

$$\mathcal{D}(P \wedge Q) \mathrel{\widehat{=}} (\mathcal{D}(P) \wedge \mathcal{D}(Q)) \vee (\mathcal{D}(P) \wedge \neg P) \vee (\mathcal{D}(Q) \wedge \neg Q) \tag{7}$$
$$\mathcal{D}(\forall x \cdot P) \mathrel{\widehat{=}} (\forall x \cdot \mathcal{D}(P)) \vee (\exists x \cdot \mathcal{D}(P) \wedge \neg P) \tag{8}$$

Intuitively, the above definitions enumerate *all* the possible conditions where a conjunctive or universally quantified predicate *could* be well-defined. From these definitions we can see that $\mathcal{D}$ is itself total and can always be eliminated from any formula.

A formal derivation of the above definitions can be found in [4] and a semantic treatment of $\mathcal{D}$ can be found in [8], [7], and [10], but for the purpose of this paper it is sufficient to accept the above syntactic equivalences as the definition of $\mathcal{D}$.

5.2 Proving Properties about $\mathcal{D}$

In this section (and in §6.4) we show some important *logical* (as opposed to syntactic) properties about $\mathcal{D}$. Care must be taken when proving statements that contain both syntactic and logical operators. Since $\mathcal{D}$ is not a logical operator, but a syntactic one, modifying its argument using standard logical transformations is not valid. For instance, given that '$P \Leftrightarrow Q$' holds (i.e. is valid in *FoPCe*), it is wrong to conclude that '$\mathcal{D}(P) \Leftrightarrow \mathcal{D}(Q)$' (consider the valid predicate '$\top \Leftrightarrow \mathit{pred}(0) = \mathit{pred}(0)$'). The only modifications that can be made to the arguments of $\mathcal{D}$ are purely syntactic ones, such as applying syntactic rewrites (using syntactic definitions that use '$\widehat{=}$'). In this section we state some properties of $\mathcal{D}$.

$\mathcal{D}$ of WD predicates. An important property of $\mathcal{D}$ is that for any formula F,

$$\mathcal{D}(\mathcal{D}(F)) \Leftrightarrow \top \tag{9}$$

This means that all WD predicates are themselves well-defined . This property can be proved by induction on the structure of basic formulæ. A proof of this nature can be found in the appendix of [4]. Note that the above property is expressed in terms of logical equivalence '$\Leftrightarrow$' and not syntactic definition '$\widehat{=}$'.

$\mathcal{D}$ for Derived Logical Operators. The following equivalences can be used to compute the WD predicates of the derived logical operators $\top$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ and $\exists$:

$$\mathcal{D}(\top) \Leftrightarrow \top \tag{10}$$

$$\mathcal{D}(P \vee Q) \Leftrightarrow (\mathcal{D}(P) \wedge \mathcal{D}(Q)) \vee (\mathcal{D}(P) \wedge P) \vee (\mathcal{D}(Q) \wedge Q) \tag{11}$$

$$\mathcal{D}(P \Rightarrow Q) \Leftrightarrow (\mathcal{D}(P) \wedge \mathcal{D}(Q)) \vee (\mathcal{D}(P) \wedge \neg P) \vee (\mathcal{D}(Q) \wedge Q) \tag{12}$$

$$\mathcal{D}(P \Leftrightarrow Q) \Leftrightarrow \mathcal{D}(P) \wedge \mathcal{D}(Q) \tag{13}$$

$$\mathcal{D}(\exists x \cdot P) \Leftrightarrow (\forall x \cdot \mathcal{D}(P)) \vee (\exists x \cdot \mathcal{D}(P) \wedge P) \tag{14}$$

The statements above can be proved using *FoPCe* (considering the discussion in §2.3 and the precautions stated in the beginning of this section) after unfolding the definitions of the derived logical operators and $\mathcal{D}$.

6 Well-Definedness and Proof

This section contains the main contribution of this paper. The theme of §5 was the well-definedness of individual formulæ. In this section we show how the notion of well-definedness can be integrated into proofs (i.e. sequents and proof rules). In §6.1 we define $\mathcal{D}$ for sequents. We then formally define the notions of a well-defined sequent (§6.2) and a WD preserving proof rule (§6.3) as motivated in §4. In §6.4 we derive the proof calculus $FoPCe_{\mathcal{D}}$, the WD preserving version of *FoPCe*, that we use to preserve well-definedness in a proof. We summarise the results of this section in §6.5.

6.1 Defining $\mathcal{D}$ for Sequents

We now extend our definition of $\mathcal{D}$ to sequents. Observing that the sequent ' $\mathsf{H} \vdash G$ ' is valid iff ' $\vdash \forall \vec{x}.\ \bigwedge \mathsf{H} \ \Rightarrow\ G$ ' is also valid (where '$\forall \vec{x}$' denotes the universal quantification of all free variables occurring in H and G, and '$\bigwedge \mathsf{H}$' denotes the conjunction of all predicates present in H), we extend our well-definedness operator to sequents as follows:

$$\mathcal{D}(\ \mathsf{H} \vdash G\ )\ \widehat{=}\ \mathcal{D}(\ \forall \vec{x}.\ \bigwedge \mathsf{H}\ \Rightarrow\ G\ ) \tag{15}$$

Note that if we blindly use the definitions (15), (8), (12), and (7) to evaluate '$\mathcal{D}(\ \mathsf{H} \vdash G\ )$' we get a disjunctive predicate that grows *exponentially* with respect to the number of free variables and hypotheses in the sequent. We present ways to overcome this problem in §6.2 and §7.

6.2 Well-Defined Sequents

In §4 we said that the initial sequent of the Validity proof obligation could be considered well-defined since we also prove WD. More generally, we say that a sequent '$\mathsf{H} \vdash G$' is well-defined if we can additionally assume '$\mathcal{D}(\ \mathsf{H} \vdash G\ )$' to be present in its hypotheses. We thereby *encode* the well-definedness of a sequent within its hypotheses. We introduce additional syntactic sugar '$\vdash_{\mathcal{D}}$' to denote such a well-defined sequent:

$$\mathsf{H} \vdash_{\mathcal{D}} G \;\widehat{=}\; \mathcal{D}(\,\mathsf{H} \vdash G\,), \mathsf{H} \;\vdash\; G \tag{16}$$

Re-stating WD and Validity. We may re-state our original proof obligations from §4 in terms of '$\vdash_{\mathcal{D}}$' as follows:

$\mathsf{WD}_{\mathcal{D}}$: $\vdash_{\mathcal{D}} \mathcal{D}(\,\mathsf{H} \vdash G\,)$	$\mathsf{Validity}_{\mathcal{D}}$: $\mathsf{H} \vdash_{\mathcal{D}} G$

Justification. The $\mathsf{WD}_{\mathcal{D}}$ proof obligation is equivalent to the original WD proof obligation since we know from (9) that '$\mathcal{D}(\mathcal{D}(\mathsf{H} \vdash G)) \Leftrightarrow \top$'. To get $\mathsf{Validity}_{\mathcal{D}}$, we add the extra hypothesis '$\mathcal{D}(\,\mathsf{H} \vdash G\,)$' to Validity using the *cut* rule whose first antecedent can be discharged using the proof of WD.

We return to the issue of proving $\mathsf{WD}_{\mathcal{D}}$ and $\mathsf{Validity}_{\mathcal{D}}$ in §7. The rest of this section is concerned with proving well-defined '$\vdash_{\mathcal{D}}$' sequents in general.

Simplifying $\vdash_{\mathcal{D}}$. Directly unfolding (16) introduces the predicate '$\mathcal{D}(\,\mathsf{H} \vdash G\,)$' that, as we have seen in §6.1, grows exponentially when further unfolded. We avoid unfolding '$\mathcal{D}(\,\mathsf{H} \vdash G\,)$' by using the following derived rule instead of (16) to introduce or eliminate $\vdash_{\mathcal{D}}$ from a proof :

$$\frac{\widehat{\mathcal{D}}(\mathsf{H}), \mathcal{D}(G), \mathsf{H} \vdash G}{\mathsf{H} \vdash_{\mathcal{D}} G} \vdash_{\mathcal{D}} eqv$$

The $\widehat{\mathcal{D}}$ operator is $\mathcal{D}$ extended for finite sets of formulæ (i.e. $\widehat{\mathcal{D}}(\mathsf{F}) \;\widehat{=}\; \bigcup_{F \in \mathsf{F}} \{\mathcal{D}(F)\}$). Note that $\widehat{\mathcal{D}}(\mathsf{H})$ denotes a set of predicates. The double inference line means that this rule can be used in both directions. The rule $\vdash_{\mathcal{D}} eqv$ says that when proving the validity of a well defined sequent, we may assume that all its hypotheses and its goal are *individually* well-defined.

Proof of $\vdash_{\mathcal{D}} eqv$. We will use the following three derived rules as lemmas in order to prove $\vdash_{\mathcal{D}} eqv$:

$$\frac{\mathcal{D}(P) \vdash P}{\mathcal{D}(\forall x \cdot P) \vdash \forall x \cdot P} \forall_{\mathcal{D}} \qquad \frac{\mathcal{D}(P), \mathcal{D}(Q), P \vdash Q}{\mathcal{D}(P \Rightarrow Q) \vdash P \Rightarrow Q} \Rightarrow_{\mathcal{D}}$$

$$\frac{\mathsf{H}, \mathcal{D}(P), \mathcal{D}(Q), P, Q \vdash R}{\mathsf{H}, \mathcal{D}(P \wedge Q), P \wedge Q \vdash R} \wedge_{\mathcal{D}}$$

The proofs of $\Rightarrow_{\mathcal{D}}$ and $\wedge_{\mathcal{D}}$ in both directions are straightforward using the definitions of $\mathcal{D}$ and the rules of *FoPCe*, and are similar to the proof of $\vdash_{\mathcal{D}} eqv_{simple}$ shown later in this section. The proof of $\forall_{\mathcal{D}}$ though is tricky, but almost identical to the proof of another derived rule $\forall goal_{\mathcal{D}}$ presented in §6.4.

The proof of $\vdash_{\mathcal{D}} eqv$ proceeds as follows. The logical content of the sequent (i.e. the hypotheses and the goal) is first *packed* into the goal of the sequent using the rules of *FoPCe*. This goal is then *unraveled* in parallel with its well-definedness predicate using the three derived rules stated above. Here is the proof:

$$\boxed{\dfrac{\widehat{\mathcal{D}}(\mathsf{H}), \mathcal{D}(G), \mathsf{H} \vdash G}{\mathsf{H} \vdash_{\mathcal{D}} G} \vdash_{\mathcal{D}} eqv}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\widehat{\mathcal{D}}(\mathsf{H}), \mathcal{D}(G), \mathsf{H} \vdash G}{\mathcal{D}(\bigwedge \mathsf{H}), \mathcal{D}(G), \bigwedge \mathsf{H} \vdash G}\ \wedge_{\mathcal{D}}^{|\mathsf{H}|}}{\mathcal{D}(\ \bigwedge \mathsf{H}\ \Rightarrow\ G\ )\ \vdash\ \bigwedge \mathsf{H}\ \Rightarrow\ G}\ \Rightarrow_{\mathcal{D}}}{\mathcal{D}(\ \forall \vec{x}\cdot\ \bigwedge \mathsf{H}\ \Rightarrow\ G\ )\ \vdash\ \forall \vec{x}\cdot\ \bigwedge \mathsf{H}\ \Rightarrow\ G}\ \forall_{\mathcal{D}}^{|\vec{x}|}}{\mathcal{D}(\ \mathsf{H} \vdash G\ )\ \vdash\ \forall \vec{x}\cdot\ \bigwedge \mathsf{H}\ \Rightarrow\ G}\ (15)}{\mathcal{D}(\ \mathsf{H} \vdash G\ ), \mathsf{H}\ \vdash\ G}\ \mathit{FoPCe}}{\mathsf{H} \vdash_{\mathcal{D}} G}\ (16)$$

In order to save space and the reader's patience, only the important steps of proofs are shown in this paper. Each step is justified using standard proof rules or previously appearing definitions and equivalences. In the proof above, superscripts above the rules $\forall_{\mathcal{D}}$ and $\wedge_{\mathcal{D}}$ indicate the number of applications of these rules. For instance $\forall_{\mathcal{D}}^{|\vec{x}|}$ indicates $|\vec{x}|$ (which is the number of free variables in the original sequent) applications of the rule $\forall_{\mathcal{D}}$. Note that this is allowed since the number of free variables ($|\vec{x}|$) and hypotheses ($|\mathsf{H}|$) contained in a sequent are finite.

In what follows we try to give the reader intuition on why $\vdash_{\mathcal{D}} eqv$ is valid since this cannot be easily seen from the proof just presented. The fact that $\vdash_{\mathcal{D}} eqv$ holds in the downward direction (i.e. if the hypotheses and goal of a sequent are well-defined, then the sequent as a whole is well-defined) is easy to see since hypotheses are weakened. For the upward direction, we present the proof of the simpler case where the well-defined sequent has no free variables and only one hypothesis. The derived rule corresponding to this simple case appears boxed on the right, followed by its proof:

$$\boxed{\dfrac{\mathcal{D}(H), \mathcal{D}(G), H \vdash G}{H \vdash_{\mathcal{D}} G} \vdash_{\mathcal{D}} eqv_{simple}}$$

$$\dfrac{\dfrac{\dfrac{\mathcal{D}(H), \mathcal{D}(G), H \vdash G \quad \dfrac{}{\mathcal{D}(H), \neg H, H \vdash G}\ \neg hyp; hyp \quad \dfrac{}{\mathcal{D}(G), G, H \vdash G}\ hyp}{(\mathcal{D}(H) \wedge \mathcal{D}(G)) \vee (\mathcal{D}(H) \wedge \neg H) \vee (\mathcal{D}(G) \wedge G), H \vdash G}\ \mathit{FoPCe}}{\mathcal{D}(H \Rightarrow G), H \vdash G}\ (12)}{H \vdash_{\mathcal{D}} G}\ (16); (15)$$

From the proof above we can see that, apart from the case where the hypothesis and the goal are individually well-defined, all other possible cases in which the sequent could be well-defined (i.e. the remaining disjuncts of '$\mathcal{D}(H \Rightarrow G)$') can be discharged using the rules in *FoPCe*.

6.3 WD Preserving Proof Rules

We say that a proof rule preserves well-definedness iff its consequent and antecedents only contain well-defined sequents (i.e. $\vdash_{\mathcal{D}}$ sequents). Examples of WD preserving proof rules can be found in §6.4.

We may derive such rules by first using $\vdash_{\mathcal{D}} eqv$ to rewrite $\vdash_{\mathcal{D}}$ sequents in terms of '$\vdash$' and then use *FoPCe* and the properties of $\mathcal{D}$ to complete the proof. Such proofs are discussed in detail in §6.4.

6.4 Deriving $FoPCe_{\mathcal{D}}$

We now have enough formal machinery in place to derive the WD preserving proof calculus $FoPCe_{\mathcal{D}}$. For each proof rule 'r' in *FoPCe* we derive its WD preserving version '$r_{\mathcal{D}}$' that only contains sequents using '$\vdash_{\mathcal{D}}$' instead of '$\vdash$'. Here are the resulting proof rules for $basicFoPCe_{\mathcal{D}}$ that are (apart from the $\vdash_{\mathcal{D}}$ turnstile) identical to their counterparts in *basicFoPCe*:

$$\frac{}{\mathsf{H}, P \vdash_{\mathcal{D}} P}\ hyp_{\mathcal{D}} \qquad \frac{\mathsf{H} \vdash_{\mathcal{D}} Q}{\mathsf{H}, P \vdash_{\mathcal{D}} Q}\ mon_{\mathcal{D}} \qquad \frac{\mathsf{H}, \neg P \vdash_{\mathcal{D}} \bot}{\mathsf{H} \vdash_{\mathcal{D}} P}\ contr_{\mathcal{D}}$$

$$\frac{}{\mathsf{H}, \bot \vdash_{\mathcal{D}} P}\ \bot hyp_{\mathcal{D}} \qquad \frac{\mathsf{H}, P \vdash_{\mathcal{D}} \bot}{\mathsf{H} \vdash_{\mathcal{D}} \neg P}\ \neg goal_{\mathcal{D}} \qquad \frac{\mathsf{H} \vdash_{\mathcal{D}} P}{\mathsf{H}, \neg P \vdash_{\mathcal{D}} Q}\ \neg hyp_{\mathcal{D}}$$

$$\frac{\mathsf{H} \vdash_{\mathcal{D}} P \qquad \mathsf{H} \vdash_{\mathcal{D}} Q}{\mathsf{H} \vdash_{\mathcal{D}} P \wedge Q}\ \wedge goal_{\mathcal{D}} \qquad \frac{\mathsf{H}, P, Q \vdash_{\mathcal{D}} R}{\mathsf{H}, P \wedge Q \vdash_{\mathcal{D}} R}\ \wedge hyp_{\mathcal{D}}$$

$$\frac{\mathsf{H} \vdash_{\mathcal{D}} P}{\mathsf{H} \vdash_{\mathcal{D}} \forall x \cdot P}\ \forall goal_{\mathcal{D}}\ (x\ \underline{\mathsf{nfin}}\ \mathsf{H})$$

$$\frac{}{\mathsf{H} \vdash_{\mathcal{D}} E = E}\ = goal_{\mathcal{D}} \qquad \frac{\mathsf{H} \vdash_{\mathcal{D}} [x := E]P}{\mathsf{H}, E = F \vdash_{\mathcal{D}} [x := F]P}\ = hyp_{\mathcal{D}}$$

The remaining rules, *cut* and $\forall hyp$, need to be reformulated by adding new antecedents (that appear boxed below) to make them WD preserving:

$$\frac{\boxed{\mathsf{H} \vdash_{\mathcal{D}} \mathcal{D}(\mathrm{P})} \qquad \mathsf{H} \vdash_{\mathcal{D}} P \qquad \mathsf{H}, P \vdash_{\mathcal{D}} Q}{\mathsf{H} \vdash_{\mathcal{D}} Q}\ cut_{\mathcal{D}}$$

$$\frac{\boxed{\mathsf{H} \vdash_{\mathcal{D}} \mathcal{D}(\mathrm{E})} \qquad \mathsf{H}, [x := E]P \vdash_{\mathcal{D}} Q}{\mathsf{H}, \forall x \cdot P \vdash_{\mathcal{D}} Q}\ \forall hyp_{\mathcal{D}}$$

These new antecedents are WD sub-goals that need to be discharged when these rules are used in a proof.

The original rules (*cut* and $\forall hyp$) are not WD preserving *since they introduce new predicates and expressions that may be ill-defined* into a proof. Note that the converse is not true. A valid proof rule in *FoPCe* that does not introduce any new formulæ into a proof can be non WD preserving. The following derived proof rule illustrates this:

$$\frac{\mathsf{H} \vdash P \qquad \mathsf{H}, P \vdash Q}{\mathsf{H} \vdash P \wedge Q}$$

Although this rule is valid (it can be proved using $\wedge goal$ and cut) and does not introduce any new formulæ, it does not preserve well-definedness. The first antecedent would be ill-defined in the case where P (say '$pred(x) = 0$') is only well-defined in conjunction with Q (say '$x \neq 0$').

The proofs of the rules of $basicFoPCe_{\mathcal{D}}$ are discussed later in §6.4.

Derived Logical Operators. Once we have derived the rules of $basicFoPCe_{\mathcal{D}}$ stated above, we may use them directly (i.e. without the detour through $\vdash$ sequents) to derive the corresponding WD preserving proof rules for the derived logical operators $\top$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ and $\exists$. The statements of these rules can be found in §4.5.3 of [17]. The only rule here that needs modification is the existential dual of $\forall hyp$ (i.e. $\exists goal$). The resulting proof rules constitute our complete WD preserving proof calculus $FoPCe_{\mathcal{D}}$.

Conditional Definitions. The payoff achieved by using $FoPCe_{\mathcal{D}}$ instead of $FoPCe$ in proofs is that conditional definitions in $FoPCe$ become 'unconditional' in $FoPCe_{\mathcal{D}}$. The '$\vdash_{\mathcal{D}}$' version of the f_{def} rule from §3 is:

$$\frac{}{\vdash_{\mathcal{D}} y = f(\vec{x}) \;\Leftrightarrow\; D^f_{\vec{x},y}}\; f_{def\,\mathcal{D}}$$

The above rule can be derived from f_{def} since $\mathcal{D}\left(\; y = f(\vec{x}) \Leftrightarrow D^f_{\vec{x},y}\right) \Rightarrow C^f_{\vec{x}}$. The above rule differs from f_{def} in that it does not explicitly require the WD condition '$C^f_{\vec{x}}$' in order to be applied. This makes proofs involving partial functions shorter and less tedious to perform. As a result, derived rules such as '$x \neq 0 \vdash pred(x + y) = pred(x) + y$' can also be applied without explicitly having to prove its premise '$x \neq 0$'.

Proofs of $basicFoPCe_{\mathcal{D}}$. The proofs of each rule in $basicFoPCe_{\mathcal{D}}$ that appear in §6.4 follow essentially from $\vdash_{\mathcal{D}} eqv$, the properties of $\mathcal{D}$, and the rules in $FoPCe$. Since some of the proofs are not straightforward we outline some of their major steps in this section as an aid the reader who wants to reproduce them. This section may otherwise be skipped.

The proofs of $hyp_{\mathcal{D}}$, $\bot hyp_{\mathcal{D}}$ and $= goal_{\mathcal{D}}$ are trivial since these rules contain no antecedents. In general, any valid rule having no antecedents is trivially WD preserving.

The proofs for $mon_{\mathcal{D}}$, $contr_{\mathcal{D}}$, $\neg goal_{\mathcal{D}}$, $\neg hyp_{\mathcal{D}}$, and $\wedge hyp_{\mathcal{D}}$ are straightforward and similar in style to the proof of $cut_{\mathcal{D}}$ shown below:

$$\boxed{\frac{\mathsf{H} \vdash_{\mathcal{D}} \mathcal{D}(P) \qquad \mathsf{H} \vdash_{\mathcal{D}} P \qquad \mathsf{H}, P \vdash_{\mathcal{D}} Q}{\mathsf{H} \vdash_{\mathcal{D}} Q}\; cut_{\mathcal{D}}}$$

$$\frac{\dfrac{\mathsf{H} \vdash_{\mathcal{D}} \mathcal{D}(P)}{\widehat{\mathcal{D}}(\mathsf{H}), \mathcal{D}(Q), \mathsf{H} \vdash \mathcal{D}(P)}\; (9); \vdash_{\mathcal{D}} eqv \qquad \dfrac{\mathsf{H} \vdash_{\mathcal{D}} P \qquad \mathsf{H}, P \vdash_{\mathcal{D}} Q}{\widehat{\mathcal{D}}(\mathsf{H}), \mathcal{D}(Q), \mathsf{H}, \mathcal{D}(P) \vdash Q}\; cut_{(P)}; \vdash_{\mathcal{D}} eqv}{\dfrac{\widehat{\mathcal{D}}(\mathsf{H}), \mathcal{D}(Q), \mathsf{H} \vdash Q}{\mathsf{H} \vdash_{\mathcal{D}} Q}\; \vdash_{\mathcal{D}} eqv}\; cut_{(\mathcal{D}(P))}$$

The proofs of $\forall hyp_{\mathcal{D}}$ and $= goal_{\mathcal{D}}$ require the following additional properties about how $\mathcal{D}$ interacts with the substitution operator:

$$\mathcal{D}([x := E]F) \Rightarrow [x := E]\mathcal{D}(F) \tag{17}$$

$$([x := E]\mathcal{D}(F)) \wedge \mathcal{D}(E) \Rightarrow \mathcal{D}([x := E]F) \tag{18}$$

Both these properties can be proved by induction on the structure of basic formulæ.

The proofs of $\wedge goal_{\mathcal{D}}$ and $\forall goal_{\mathcal{D}}$ are tricky and require rewriting definitions (7) and (8) as follows:

$$\mathcal{D}(P \wedge Q) \Leftrightarrow (\mathcal{D}(P) \wedge (P \Rightarrow \mathcal{D}(Q))) \ \vee \ (\mathcal{D}(Q) \wedge (Q \Rightarrow \mathcal{D}(P))) \tag{19}$$

$$\mathcal{D}(\forall x \cdot P) \Leftrightarrow \exists x \cdot \ (\mathcal{D}(P) \wedge (P \Rightarrow \forall x \cdot \mathcal{D}(P))) \tag{20}$$

Using these equivalences instead of (7) and (8) allows for more natural case splits in these proofs. The proof of $\wedge goal_{\mathcal{D}}$ is similar to the proof of $\forall goal_{\mathcal{D}}$ shown below:

$$\boxed{\frac{\mathsf{H} \vdash_{\mathcal{D}} P}{\mathsf{H} \vdash_{\mathcal{D}} \forall x \cdot P} \ \forall goal_{\mathcal{D}} \ (x \ \underline{\mathsf{nfin}} \ \mathsf{H})}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{H} \vdash_{\mathcal{D}} P}{\widehat{\mathcal{D}}(\mathsf{H}), \mathcal{D}(P), \mathsf{H} \vdash P} \vdash_{\mathcal{D}} eqv \quad \dfrac{\dfrac{\mathsf{H} \vdash_{\mathcal{D}} P}{\widehat{\mathcal{D}}(\mathsf{H}), \forall x \cdot \mathcal{D}(P), \mathsf{H} \vdash P} \forall hyp; \vdash_{\mathcal{D}} eqv}{\widehat{\mathcal{D}}(\mathsf{H}), \forall x \cdot \mathcal{D}(P), \mathsf{H} \vdash \forall x \cdot P} \forall goal \ (x \ \underline{\mathsf{nfin}} \ \mathsf{H})}{\widehat{\mathcal{D}}(\mathsf{H}), \mathcal{D}(P), \forall x \cdot \mathcal{D}(P), \mathsf{H} \vdash \forall x \cdot P} mon_{(\mathcal{D}(P))}}{\widehat{\mathcal{D}}(\mathsf{H}), \mathcal{D}(P), P \Rightarrow \forall x \cdot \mathcal{D}(P), \mathsf{H} \vdash \forall x \cdot P} \Rightarrow hyp}{\widehat{\mathcal{D}}(\mathsf{H}), \exists x \cdot (\mathcal{D}(P) \wedge (P \Rightarrow \forall x \cdot \mathcal{D}(P))), \mathsf{H} \vdash \forall x \cdot P} \exists hyp; \wedge hyp}{\mathsf{H} \vdash_{\mathcal{D}} \forall x \cdot P} \vdash_{\mathcal{D}} eqv; (20)$$

Note that the rules $\Rightarrow hyp$ and $\exists hyp$ above are the standard left hand sequent calculus rules for implication and existential quantification. We now summarise the results of this section.

6.5 Summary

In this section we have shown how the notion of well-definedness can be integrated into proofs by extending the definition of $\mathcal{D}$ to sequents (§6.1), and characterising well-defined '$\vdash_{\mathcal{D}}$' sequents (§6.2). We have derived the proof rule $\vdash_{\mathcal{D}} eqv$ (§6.2) that allows us to freely move between ordinary '$\vdash$' sequents and well-defined '$\vdash_{\mathcal{D}}$' sequents in proofs. We have formally derived the proof calculus $FoPCe_{\mathcal{D}}$ (§6) whose rules preserve well-definedness. The rules of $FoPCe_{\mathcal{D}}$ are identical to those in $FoPCe$ except for three cases (cut, $\forall hyp$ and its dual $\exists goal$) where additional WD sub-goals need to be proved.

We now return to the practical issue of proving the WD and Validity proof obligations introduced in §4.

7 Proving $\mathsf{WD}_{\mathcal{D}}$ and $\mathsf{Validity}_{\mathcal{D}}$

In §6.2 we re-stated our original proof obligations from §4 in terms of '$\vdash_{\mathcal{D}}$' as follows:

$\mathsf{WD}_{\mathcal{D}}$: $\vdash_{\mathcal{D}} \mathcal{D}(\ \mathsf{H} \vdash G\ )$	$\mathsf{Validity}_{\mathcal{D}}$: $\mathsf{H} \vdash_{\mathcal{D}} G$

Proving $\mathsf{WD}_{\mathcal{D}}$. In our practical setting we factor out proving $\mathsf{WD}_{\mathcal{D}}$ for each proof obligation *individually* by proving instead that the (source) models used to generate these proof obligations are well-defined. Details on well-definedness of models can be found in [7]. We are guaranteed that all proof obligations generated from a well-defined model are themselves well-defined and therefore do not need to generate or prove the well-definedness of each proof obligation individually. This considerably reduces the number of proofs that need to be done.

Proving $\mathsf{Validity}_{\mathcal{D}}$. From the '$\vdash_{\mathcal{D}}$' turnstile we immediately see that the initial sequent of $\mathsf{Validity}_{\mathcal{D}}$ is well-defined. We have two choices for how to proceed with this proof. We may either use $FoPCe_{\mathcal{D}}$ to preserve well-definedness, or the standard *FoPCe*.

We prefer using the WD preserving calculus $FoPCe_{\mathcal{D}}$ (with additional WD sub-goals) instead of *FoPCe* for *interactive* proofs for three reasons. Firstly, as seen in §4, the assumption that a sequent is well-defined can be used to greatly ease its proof. Secondly, the extra WD sub-goals require only minimal additional effort to prove in practice, and are in most cases automatically discharged. Thirdly, proving WD sub-goals allows us to *filter out* erroneous ill-defined formulæ entered by the user.

Alternatively, we may use the rule $\vdash_{\mathcal{D}} eqv$ (§6.2) to make the well-definedness assumptions of a '$\vdash_{\mathcal{D}}$' sequent explicit and use the standard proof rules in *FoPCe* to complete a proof. This may not be a prudent way to perform interactive proof, but it allows us to use existing automated theorem provers for *FoPCe* (that do not have any notion of well-definedness) to automatically discharge pending sub-goals, for which we have observed favourable results.

8 Related Work

A lot of work has been done in the area of reasoning in the presence of partial functions. A good review of this work can be found in [13] and [4]. In this section we first describe some approaches that are most relevant to the work presented in this paper and then compare our work to these approaches in §8.1.

The current approaches to reason about the undefined can be classified into two broad categories: those that *explicitly reason* about undefined values using a three-valued logic [16], and those that *avoid* reasoning about the undefined using underspecification [13]. We start with the former.

A well known approach is the Logic of Partial Functions (LPF) [15,6] used by the VDM [14] community. Its semantics is based on three-valued logic [16]. The

resulting proof calculus for LPF can then be used to *simultaneously* prove the validity and well-definedness of a logical statement. A drawback of using LPF (or any other special-purpose proof calculus) is that it differs from the standard predicate calculus since it disallows use of the law of excluded middle (to avoid proving '$3/0 = x \vee 3/0 \neq x$') and additionally requires a second 'weak' notion of equality (to avoid proving '$3/0 = 3/0$'). Additional effort is therefore needed to learn or automate LPF, as for any special-purpose proof calculus, as mentioned in §1.

In PVS [19], partial functions are modelled as total functions whose domain is a predicate subtype. For instance, the partial function '/' is defined as a total function whose second argument belongs to the subtype of non-zero reals. Type-checking then avoids ill-definedness but requires proof. The user needs to prove type correctness conditions (TCCs) before starting or introducing new formulæ into a proof. A shortcoming of this approach is that type-checking requires complicated typing rules [20] and special tool support. This approach additionally blurs the distinction between type-checking (which is usually accepted to be automatically decidable) and proof.

In [7], Behm et al. use a three-valued semantics to develop a proof calculus for B [2]. Its main difference from LPF is that the undefined value, although part of the logical semantics, does not enter into proofs, as explained below. In this approach, all formulæ that appear in a proof need to be proved to be well-defined. Proving well-definedness is similar to proving TCCs in PVS. It has the role of *filtering out* expressions that may be ill-defined. Once this is done, the proof may continue in a *pseudo-two-valued* logic since the undefined value is proved never to occur. The drawback of this approach is similar to that of LPF. Although the proof calculus presented for this pseudo-two-valued logic "is close to the standard sequent calculus"[7], this too is a special-purpose logic. No concrete connection with the standard predicate calculus is evident since this approach, from the start, assumes a three-valued semantics.

In [4], Abrial and Mussat formalise the notion of well-definedness without any detour through a three-valued semantics, remaining entirely within the "syntactic manipulation of proofs"[4] in standard predicate calculus. The resulting well-definedness filter is identical to that in [7]. They formally show how proving statements that passed this filter could be made simpler (i.e. with fewer checks) on the basis of their well-definedness. What is missing in [4] however is a proof calculus (like the one in [7]) that preserves well-definedness, which could additionally be used for interactive proof.

In [8], Berezin et al. also use the approach of filtering out ill-defined statements before attempting to prove them in the automated theorem prover CVC lite. The filter used is identical to the one used in [7] and [4]. Although they too start from a three-valued logic, they show (using semantic arguments) how the proof of a statement that has passed this filter may proceed in standard two-valued logic. Apart from introducing three-valued logic only to reduce it later to two-valued logic, this approach is concerned with purely automated theorem proving and therefore provides no proof calculus that preserves well-definedness to use in

interactive proofs. It is advantageous to preserve well-definedness in interactive proofs (reasons for this are given in §4).

The idea of *avoiding* reasoning about undefined values using underspecification [13] is used in many approaches that stick to using two-valued logic in the presence of partial functions. This is the approach used in Isabelle/HOL [18], HOL [12], JML [9] and Spec# [5]. In this setting, an expression such as '$3/0$' is still a valid expression, but its value is unspecified. Although underspecification allows proofs involving partial functions to be done in two-valued logic, it has two shortcomings. First, (as described in §3) it also allows statements that contain ill-defined terms such as '$3/0 = 3/0$' to be proved valid. In the context of generating or verifying program code, expressions such as '$3/0$' originating from a proved development can lead to a run-time error as described in [9]. Second, doing proofs in this setting may require *repeatedly* proving that a possibly undefined expression (e.g. '$3/x$') is actually well defined (i.e. that '$x \neq 0$') in multiple proof steps.

8.1 Comparison

In this section we compare the approach presented in this paper (§5, §6) with the related work just presented. The work presented here extends the approach of [4] by showing how the notion of well-definedness can be integrated into a proof calculus that is suitable for interactive proof.

The role of proving TCCs in PVS is identical to that of proving well-definedness in our approach (i.e. proving the WD proof obligation and the additional WD sub-goals in $cut_{\mathcal{D}}$, $\forall hyp_{\mathcal{D}}$, and $\exists goal_{\mathcal{D}}$). With regards to its logical foundations, we find the possibility of directly defining *truly* partial functions in our setting more convenient and intuitive as opposed to expressing them as total functions over a restricted subtype. The logical machinery we use is much simpler too since we do not need to introduce predicate subtypes and dependent types for this purpose. Since we use standard (decidable) type-checking we have a clear conceptual separation between type-checking and proof. Although our approach does not eliminate the undecidability of checking well-definedness, it saves type checking from being undecidable.

With respect to B [7] and LPF [6], the approach used here does not start from a three-valued semantics but instead reduces all reasoning about well-definedness to standard predicate calculus. We develop the notion of well-definedness purely on the basis of the syntactic operator '$\mathcal{D}$' and proofs in standard predicate calculus. In §6 we derive a proof calculus preserving well-definedness that is identical to the one presented in [7]. Alternatively, we could have chosen to derive the proof calculus used in LPF in a similar fashion. If our definition of well-defined sequents is modified from the one appearing in §6.2 to

$$\mathsf{H} \vdash_{\scriptscriptstyle LPF} G \;\widehat{=}\; \widehat{\mathcal{D}}(\mathsf{H}), \mathsf{H} \vdash G \wedge \mathcal{D}(G) \tag{21}$$

the rules that follow for a proof calculus that preserves '$\vdash_{\scriptscriptstyle LPF}$' correspond to those in LPF [6]. The only difference between '$\vdash_{\scriptscriptstyle LPF}$' and '$\vdash_{\mathcal{D}}$' is that the latter also assumes the well-definedness of the goal, whereas this has to be additionally proved

for '$\vdash_{LPF}$'. We therefore have a clear basis to compare these two approaches. In LPF, well-definedness and validity of the goal are proved *simultaneously*, whereas in [7] (and also as presented in §4), these two proofs are performed separately, where proving WD acts as a *filter*. Since what is proved is essentially the same, the choice of which approach to use is a methodological preference. We use the latter approach although it requires proving two proof obligations because of four reasons. First, the majority of the WD sub-goals that we encounter in practice (from models and interactive proof steps) are discharged automatically. Second, failure to discharge a proof obligation due to ill-definedness can be detected earlier and more precisely, before effort is spent on proving validity. Third, the structure of '$\vdash_{\mathcal{D}}$' sequents allows us to more directly use the results in [4] and [8] to automate proofs. Fourth, we find $FoPCe_{\mathcal{D}}$ more intuitive to use in interactive proofs since its rules are 'closer' to the standard sequent calculus (only three rules need to be modified with an extra WD sub-goal).

An additional contribution over [7] (and [6]) is that we may, at any time, choose to reduce all our reasoning to standard predicate calculus (using the $\vdash_{\mathcal{D}} eqv$ rule derived in §6.2). This is a choice that could not be taken in [7].

We now compare our work to related approaches that use underspecification [13]. As described in §3, underspecification is the starting point from which we develop our approach. The work presented in this paper can be used to overcome two of the shortcomings the underspecification approach mentioned earlier. First, (as discussed in §4) proving the well-definedness of proof obligations (or of the source model) gives us an additional guarantee that partial (underspecified) functions are not evaluated outside their domain in specifications or program code. This has been recently done along similar lines for Spec# [5] in [21]. Second, (as discussed in §4 and §6.3) preserving well-definedness in proofs allows us to avoid having to prove well-definedness repeatedly, every time we are confronted with a possibly ill-defined expression during proof.

9 Conclusion

In this paper we have shown how standard predicate calculus can be used to reason in a setting with potentially ill-defined expressions by extending it with new syntax and derived rules for this purpose.

The results presented in §6 provide a deeper understanding of reasoning in the context of well-definedness, and its connection with the standard predicate calculus. This work has also resulted in reducing the proof burden in the partial setting by providing better tool support within the RODIN development environment [1] for Event-B since:

- Sequents contain less hypotheses because all well-definedness hypotheses are implicit in well-defined sequents, as described in §6.2.
- Conditional definitions become unconditional as described in §6.4.
- Derived rules contain less preconditions as discussed in §4 and §6.4.
- All proofs can be reduced to proofs in standard two-valued predicate calculus as seen from the $\vdash_{\mathcal{D}} eqv$ rule in §6.2 and discussed in §7.

Acknowledgements

The author would like to thank Jean-Raymond Abrial, Cliff Jones, John Fitzgerald, David Basin, Laurent Voisin, Adam Darvas and Vijay D'silva for their comments and lively discussions.

References

1. Rigorous Open Development Environment for Complex Systems (RODIN) official website, `http://www.event-b.org/`
2. Abrial, J.-R.: The B-Book: Assigning programs to meanings. Cambridge (1996)
3. Abrial, J.-R.: Modeling in Event B: System and Softtware Design. Cambridge (to appear, 2007)
4. Abrial, J.-R., Mussat, L.: On using conditional definitions in formal theories. In: Bert, D., Bowen, J., Henson, M., Robinson, K. (eds.) B 2002 and ZB 2002. LNCS, vol. 2272, pp. 242–269. Springer, Heidelberg (2002)
5. Barnett, M., Leino, K.R.M., Schulte, W.: The Spec# programming system: An overview. In: Construction and Analysis of Safe, Secure, and Interoperable Smart Devices, pp. 49–69 (2005)
6. Barringer, H., Cheng, J.H., Jones, C.B.: A logic covering undefinedness in program proofs. Acta Inf. 21, 251–269 (1984)
7. Behm, P., Burdy, L., Meynadier, J.-M.: Well defined B. In: Bert, D. (ed.) B 1998. LNCS, vol. 1393, pp. 29–45. Springer, Heidelberg (1998)
8. Berezin, S., Barrett, C., Shikanian, I., Chechik, M., Gurfinkel, A., Dill, D.L.: A practical approach to partial functions in CVC Lite
9. Chalin, P.: Logical foundations of program assertions: What do practitioners want? In: SEFM, pp. 383–393 (2005)
10. Darvas, Á., Mehta, F., Rudich, A.: Efficient well-definedness checking. In: International Joint Conference on Automated Reasoning (IJCAR). LNCS. Springer, Heidelberg (to appear, 2008)
11. Fitzgerald, J.S., Jones, C.B.: The connection between two ways of reasoning about partial functions. Technical Report CS-TR-1044, School of Computing Science. Newcastle University (August 2007)
12. Gordon, M.J.C., Melham, T.F.: Introduction to HOL: a theorem proving environment for higher order logic. Cambridge University Press, New York (1993)
13. Gries, D., Schneider, F.B.: Avoiding the undefined by underspecification. In: van Leeuwen, J. (ed.) Computer Science Today. LNCS, vol. 1000, pp. 366–373. Springer, Heidelberg (1995)
14. Jones, C.B.: Systematic software development using VDM, 2nd edn. Prentice-Hall, Inc., Englewood Cliffs (1990)
15. Jones, C.B.: Reasoning about partial functions in the formal development of programs. Electr. Notes Theor. Comput. Sci. 145, 3–25 (2006)
16. Kleene, S.C.: Introduction to metamathematics. Bibl. Matematica. North-Holland, Amsterdam (1952)
17. Mehta, F.D.: Proofs for the Working Engineer. PhD thesis, ETH Zurich (2008)
18. Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002)
19. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system, January 15 (2001)

20. Owre, S., Shankar, N.: The formal semantics of PVS (March 1999), http://www.csl.sri.com/papers/csl-97-2/
21. Rudich, A., Darvas, Á., Müller, P.: Checking well-formedness of pure-method specifications. In: Formal Methods (FM). LNCS. Springer, Heidelberg (2008)
22. Sutcliffe, G., Suttner, C.B.: The TPTP (Thousands of Problems for Theorem Provers) Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning 21(2), 177–203 (1998)