Science of Computer Programming 94 (2014) 238–252

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Revising basic theorem proving algorithms to cope with the logic of partial functions

Cliff B. Jones *, Matthew J. Lovert, L. Jason Steggles

School of Computing Science, Newcastle University, NE1 7RU, UK

HIGHLIGHTS

- Addresses the task of modifying proof procedures to handle the Logic of Partial Functions.
- Re-uses as much as possible of the standard literature.
- Provides proofs of the things that have to be modified.

ARTICLE INFO

Article history:
Received 17 February 2013
Received in revised form 16 July 2013
Accepted 18 September 2013
Available online 2 October 2013

Keywords:
Proof
Theorem proving systems
Partial functions
LPF

ABSTRACT

Partial terms are those that can fail to denote a value; such terms arise frequently in the specification and development of programs. Earlier papers describe and argue for the use of the non-classical "Logic of Partial Functions" (LPF) to facilitate sound and convenient reasoning about such terms. This paper reviews the fundamental theorem proving algorithms — such as resolution — and identifies where they need revision to cope with LPF. Particular care is needed with "refutation" procedures. The modified algorithms are justified with respect to a semantic model. Indications are provided of further work which could lead to efficient support for LPF.

1. Introduction

Within logical expressions, terms can fail to denote proper values and as a result logical formulae involving such terms may not denote Booleans [18,24,30]. Such partial terms arise frequently — for example when applying recursive functions — in the specification of computer programs; more tellingly, reasoning about such terms is required when discharging the proof obligations generated in both establishing consistency of specifications at any level of abstraction and for justifying development steps between levels (such proof obligations can be very large for industrial applications). This raises the question of how one can reason about such formulae. Numerous approaches have been conceived over the years — most are documented in [8,12,9,10,13,16,1,14,34].

The issue of reasoning about partial functions is by no means purely theoretical: in [35], it is identified as a significant source of inconsistencies in the theorem provers for Event-B; after the 2011 "Landin seminar" by one of the current authors, David Crocker pointed out that one of very few inconsistencies in "Perfect Developer" again revolved around the issue of undefined terms.

This paper is intended to interest computer scientists in alternatives to bending partial functions into the classical model of first-order predicate calculus. It is not so much aimed at logicians.

* Corresponding author.
E-mail addresses: cliff.jones@ncl.ac.uk (C.B. Jones), matthew.lovert@ncl.ac.uk (M.J. Lovert), l.j.steggles@ncl.ac.uk (L.J. Steggles).

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.09.007

The issue of non-denoting terms can be exemplified by the following property using integer division:

$$\forall i : \mathbb{Z} \cdot (i \div i = 1) \vee \big((i-1) \div (i-1) = 1\big) \tag{1}$$

When i has the value 0, the first disjunct fails to denote a value; similarly, the second disjunct fails to denote a value when i has the value 1. The best way of thinking about the issue is to see that there is a "gap" in the denotation of the integer division operator (this view is formalised in Section 3).[1] It is however convenient to illustrate the difficulties by writing $\bot_{\mathbb{Z}}$ to stand for a missing integer value (and $\bot_{\mathbb{B}}$ for a missing Boolean value). The validity of property (1) relies on the truth of disjunctions such as $(1 \div 1 = 1) \vee (0 \div 0 = 1)$, which reduces to $(1 = 1) \vee (\bot_{\mathbb{Z}} = 1)$. With strict (weak/computational) equality (undefined if either operand is undefined), this further reduces to $true \vee \bot_{\mathbb{B}}$ which makes no sense in classical logic since its truth tables only define the propositional operators for proper Boolean values.

The approach that the current authors take to reasoning about logical formulae that include partial terms is to employ a *non-classical logic* known as the *Logic of Partial Functions* (*LPF*) [3,8,10,23,24,20], where "gaps" are handled by lifting the logical operators. Property (1) is true in LPF and its proof presents no difficulty (after some explanation, this proof is given in Fig. 2). However, property (1) can cause "issues" in other approaches to coping with non-denoting terms — for example, with McCarthy's conditional version[2] of the logical operators [29], where disjunctions and conjunctions are not commutative and quantifiers are problematic with respect to undefined values.

However, the availability of a large body of proof techniques for classical logic presents an argument against the adoption of LPF. These fundamental automated proof techniques are the foundation on which many advanced automated proof techniques are built and as such represent a natural starting point for considering the development of proof support for LPF. Determining how to modify these proof procedures for LPF and analysing the associated performance issues provides key insights into mechanising proof for a logic like LPF. Furthermore, it provides the essential foundation to facilitate the modification of more advanced proof techniques for LPF. The main contribution of this paper (Section 5) is to pinpoint the issues that arise for the adaption of techniques such as proof by refutation and resolution to cope with LPF. In some cases, the justification of the extended algorithms is essentially the same as with their classical counterparts; only where there are significant changes are new proofs provided. In particular, the soundness of the modified resolution procedure is proved; resolution completeness is the subject of on-going research.

Structure of the paper: Section 2 provides an introduction to LPF. Section 3 provides a semantics for the LPF version of the Predicate Calculus — the rest of the paper is grounded on this semantic model. Section 4 discusses normal forms. Section 5 outlines the issues present — and the changes required — for the proof procedures to cover LPF. Finally, Section 6 provides some conclusions and an indication of further work.

2. An introduction to LPF

LPF is a first order logic that can handle non-denoting logical values that arise from terms that apply partial functions and operators; it is the logic that underlies the *Vienna Development Method* (*VDM*) [18,5,15]; there was an instantiation of LPF on the *mural* formal development support system [19]. Arguments for the use of LPF are documented in several of the previously cited references, particularly [10,24,20].

It is straightforward to lift the standard two-valued truth tables for propositional operators to cover logical values that may fail to denote (see for example [26, §64]). Such tables provide the strongest possible *monotonic* extension of the familiar classical propositional operators with respect to the ordering on truth values: $\bot_{\mathbb{B}} \preceq \mathrm{tt}$ and $\bot_{\mathbb{B}} \preceq \mathrm{ff}$. As an example, the truth table for disjunction is given in Fig. 1(a). Alternatively, such truth tables can be viewed as describing a parallel (lazy) evaluation of the operands that delivers a result as soon as enough information is available; such a result would not be contradicted if a $\bot_{\mathbb{B}}$ were evaluated to a proper Boolean value.

The way in which non-denoting values can be "caught" by these extended propositional operators can be depicted as follows[3]:

$$\forall i : \mathbb{Z} \cdot \overbrace{\underbrace{(\underbrace{i \div i}_{\in \mathbb{Z}_\bot} = 1)}_{\in \mathbb{B}_\bot} \vee \underbrace{\big(\underbrace{(i-1) \div (i-1)}_{\in \mathbb{Z}_\bot} = 1\big)}_{\in \mathbb{B}_\bot}}^{\in \mathbb{B}}$$

where $\mathbb{Z}_\bot$ stands for $\mathbb{Z} \cup \{\bot_{\mathbb{Z}}\}$ and $\mathbb{B}_\bot$ stands for $\mathbb{B} \cup \{\bot_{\mathbb{B}}\}$.

[1] As explained in earlier papers, the problem of non-denoting terms is pervasive and most of these papers have used examples with recursive functions; in this paper, the fact that division is a partial operator is used to present the essential points with a minimum of extra machinery.

[2] McCarthy defined, for example, the disjunction of e_1, e_2 as **if** e_1 **then** tt **else** e_2 and referred to the first variable in such conditional expressions as the "inevitable variable" since conditionals are strict in their first argument. This interpretation is implemented in some programming languages — sometimes with distinct keywords — and this imposes a proof obligation when copying LPF expressions into program texts cf. [18].

[3] Comparisons of several differing approaches to handling undefined values are supported by pictures of this style in [21]. Note that it is *not* claimed that such types are syntactically decidable.